



Technische Universität München
TUM School of Computation, Information and Technology

Efficient Measures for Processing and Exploring Large-Scale Data Domains

Christian Reinbold

Vollständiger Abdruck der von der TUM School of Computation, Information and
Technology der Technischen Universität München zur Erlangung des akademischen
Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Alfons Kemper
Prüfer der Dissertation: 1. Prof. Dr. Rüdiger Westermann
2. Asst. Prof. Dr. Jens Schneider

Die Dissertation wurde am 27.06.2022 bei der Technischen Universität München
eingereicht und durch die Fakultät für Informatik am 21.09.2022 angenommen.

For peace

Abstract

Simulations of chaotic systems such as fluids and meteorological processes, as well as rendering applications in computer graphics, rely on accurate and thus highly resolved data representations to converge to a realistic output. Commonly, data is discretized into hundreds and thousands of samples per dimension, giving rise to giga- and terabyte-scale datasets in the case of spatial (3D) and spatiotemporal (4D) domains. Apart from that, large-scale datasets significantly gained in importance with recent advances in data-driven designs, more specifically, deep learning. Data passed to and generated by neural networks easily live in domains with millions of dimensions, of which many may or may not carry significant information. In general, large-scale datasets pose a couple of challenges that need to be addressed by the scientific community.

First, algorithms processing large-scale data have to be designed with scalability in mind. Because of the ever-increasing compute-capabilities of modern hardware, datasets reached a size where a complexity of $\mathcal{O}(n \log n)$ introduced by many acceleration structures may already be too large to either fit the structure into memory or process it in a reasonable time. For instance, we consider summed-volume tables (SVTs), a data structure that allows to integrate large-scale data over arbitrary, axis-aligned subregions in $\mathcal{O}(1)$ complexity, at a memory cost of $\mathcal{O}(n \log n)$. SVTs facilitate the creation of spatial search structures and enable interactive queries of statistical quantities over large, user-defined regions. We introduce a novel, hierarchical approach to compute and store SVTs by recursively splitting blocks of data before computing their prefix sums. The proposed splitting algorithm adapts to any prescribed memory budget between $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ by flexibly trading the number of data-fetch operations for memory. At the cost of just a couple of additional data-fetch operations per query, SVTs now can be deployed in environments where their memory footprint otherwise would not allow to do so.

Second, large-scale data—especially high-dimensional data—usually contains a lot of redundant information. It can be inferred from more compact representations that preserve as much information from the original data as possible. Thus, effective measures for identifying low-dimensional data

embeddings are required. Popular dimensionality reduction (DR) techniques such as multidimensional scaling, t-distributed stochastic neighbour embedding and uniform manifold approximation and projection address this need, but produce different data embeddings for each run. Therefore, to obtain reliable insights into the data, multiple embeddings have to be investigated and compared to each other. We present a visualization technique based on k -order Voronoi diagrams to assess the stability of ensembles—i. e. collections—of point embeddings. By partitioning the background of a mapping via a Voronoi diagram and coloring the respective cells, the user can visually identify the neighborhood relations of an embedding that also persist in most of the other ensemble members. Further, we introduce a pairwise similarity measure between embeddings that is successfully used to identify clusters and representative members in the ensemble.

In addition to classical DR techniques, task-orientated encodings of high-dimensional data have been proposed by learning low-dimensional feature spaces in neural networks. We utilize this approach in the context of computer graphics. In level of detail applications, multiple representations of the same scene at various resolutions are rendered depending on the distance to the camera. However, the validity of representations does not only depend on camera distance but view direction as well, since parts of the geometry may be occluded and do not contribute to the rendering. We approach this issue by utilizing an encoder/decoder-based neural network architecture and learning compact shape and appearance representations of high-dimensional geometry patches. They carry sufficient information to identify view directions from which geometry is not correctly represented. We utilize the network for adaptive super-sampling in ray-tracing to predict super-sampling patterns when seeing coarse-scale geometry. By having access to scene geometry information, we can identify wrongly shaded pixels due to level of detail rendering more consistently than screen-space-based alternatives.

Another challenge arises from high-dimensional latent feature spaces in deep learning lacking interpretability. To increase trust in decisions of deep neural networks, methods have to be developed to better understand the information encoded in latent features. Activation maximization facilitates interpretability through associating visual and interpretable concepts with various features. Recently, activation maximization has been used to define a taxonomy of feature classes that form across models and tasks. However, these findings were only possible by manually comparing and matching a vast amount of feature visualization. We make a first step towards automating this process, by proposing the problem of inverse feature visualization and showing that our novel method called gradient-based inverse feature visualization can identify the neurons for which a feature visualization has been generated.

Zusammenfassung

Simulationen chaotischer Systeme in der Fluidmechanik oder Meteorologie, sowie Renderings in der Computergrafik sind auf hoch aufgelöste Datenrepräsentationen angewiesen, um gegen ein realistisches Ergebnis zu konvergieren. Übliche Auflösungen bewegen sich im Bereich von hunderten und tausenden Datenpunkten in jeder Dimension, sodass im Fall räumlicher (3D) beziehungsweise raumzeitlicher (4D) Daten Größenordnungen im Bereich von Giga- und Terabytes erreicht werden. Darüber hinaus haben großskalige Datensätze durch neue Fortschritte im datengetriebenen Algorithmenentwurf, konkret Deep Learning, wesentlich an Bedeutung gewonnen. Von neuronalen Netzen generierte und verarbeitete Daten sind in Räume mit Millionen von Dimensionen mit variierendem Informationsgehalt eingebettet. Die Verarbeitung solch großskaliger Datensätze birgt eine Reihe von Herausforderungen, die es zu adressieren gilt.

Zunächst müssen Algorithmen für die Verarbeitung großskaliger Datensätze in Hinblick auf Skalierbarkeit entwickelt werden. Aufgrund der kontinuierlich steigenden Leistungsfähigkeit moderner Hardware haben Datensätze eine Größe erreicht, bei der die für viele Beschleunigungsstrukturen gängige Komplexität von $\mathcal{O}(n \log n)$ bereits zu groß ist, um Strukturen im Speicher zu halten oder in annehmbarer Zeit zu verarbeiten. In unserer Arbeit betrachten wir Summed-Volume Tables (SVTs) – eine Datenstruktur mit $\mathcal{O}(n \log n)$ Speicherbedarf – die das Integrieren der Daten über beliebige, achsenparallele Regionen in $\mathcal{O}(1)$ ermöglicht. SVTs werden für die Erzeugung räumlicher Suchstrukturen eingesetzt, und ermöglichen das interaktive Abfragen statistischer Größen über große, vom Nutzer ausgewählte Regionen. Wir präsentieren einen hierarchischen Ansatz zur Berechnung und Speicherung von SVTs, der auf der rekursiven Aufspaltung von Datenblöcken basiert. Abhängig von dem verfügbaren Speicherbudget zwischen $\mathcal{O}(n)$ und $\mathcal{O}(n \log n)$ wird eine geeignete Spaltstruktur abgeleitet, die die Anzahl benötigter Speicherzugriffe und Speicherbedarf gegeneinander abwägt. Indem einige wenige, zusätzliche Speicherzugriffe pro Abfrage erlaubt werden, können SVTs in Umgebungen eingesetzt werden, wo ihr zusätzlicher Speicherbedarf dies sonst verhindern würde.

Des Weiteren enthalten – insbesondere hochdimensionale – Daten für gewöhnlich in einem hohen Maß redundante Information. Diese kann aus kompakteren Datenrepräsentationen mit gleichem Informationsgehalt rekonstruiert werden. Daher benötigt es effektive Maßnahmen zur Identifikation niedrigdimensionaler Einbettungen von Daten. Weit verbreitete Techniken zur Dimensionsreduktion (DR) wie Multi-Dimensional Scaling, t-Distributed Stochastic Neighbour Embedding und Uniform Manifold Approximation and Projection sind hierfür geeignet, generieren jedoch in jedem Durchlauf andere 2D Einbettungen. Daher müssen mehrere Einbettungen untersucht und miteinander verglichen werden, um gesicherte Einblicke in Datensätze zu erhalten. Wir stellen eine auf Voronoi Diagrammen höherer Ordnung basierende Visualisierungstechnik vor, die es erlaubt, die Stabilität von Ensembles über 2D Einbettungen zu beurteilen. Durch Zerlegung und Einfärbung der 2D Ebene auf Basis eines Voronoi Diagramms werden Nachbarschaftsbeziehungen hervorgehoben, die ebenso in den meisten anderen Einbettungen des Ensembles zutage treten. Des Weiteren stellen wir ein neues Ähnlichkeitsmaß vor, mithilfe dessen wir Cluster sowie geeignete Repräsentanten im Ensemble identifizieren.

Neben klassischen DR-Techniken werden aufgabenorientierte Kodierungen hochdimensionaler Daten entwickelt, die durch das Erlernen von niedrigdimensionalen Merkmalsräumen in neuronalen Netzen entstehen. Wir nutzen diesen Ansatz im Kontext der Computergrafik. In klassischen Level of Detail Anwendungen werden mehrere Repräsentationen derselben Szene in verschiedenen Auflösungsstufen – abhängig von der Distanz zur Kamera – gerendert. Allerdings hängt die korrekte Wahl der Auflösungsstufe in der Praxis nicht nur von der Kameradistanz ab, sondern auch von der Blickrichtung. Je nach Perspektive können unterschiedliche Teile der Geometrie verschattet werden und tragen somit nicht zum finalen Rendering bei. Wir behandeln diesen Aspekt mithilfe einer Encoder/Decoder Netzwerkarchitektur, und lernen eine kompakte Repräsentation hochdimensionaler Geometrieausschnitte. Die gelernte Repräsentation enthält hinreichend viel Information, um Blickpunkte zu identifizieren, unter denen die Geometrie nicht korrekt durch die gewählte Auflösungsstufe dargestellt wird. Wir verwenden das Netzwerk, um auf Basis niedrig aufgelöster Geometriestrukturen geeignete Super-Sampling Strukturen im Ray-Tracing zu detektieren. Durch die Verwendung von Geometriedaten können wir falsch beleuchtete Pixel zuverlässiger detektieren als alternative, bildbasierte Ansätze.

Eine weitere Herausforderung entsteht durch die mangelnde Interpretierbarkeit hochdimensionaler Merkmalsräume wie sie im Deep Learning auftreten. Um Vertrauen in netzbasierte Entscheidungsprozesse zu schaffen, sind Methoden zur Darstellung der in den Merkmalsräumen kodierten Information unabdingbar. Aktivierungsmaximierung fördert hierbei das Verständnis durch die Generierung visueller, interpretierbarer Abbildungen von Merkmalen. Mithilfe von Aktivierungsmaximierung konnte eine Taxonomie von modell- und aufgabenübergreifenden Merkmalen entwickelt werden. Hierzu wurden in aufwändiger Handarbeit eine Vielzahl an Merkmalsabbildungen

miteinander abgeglichen. Wir schlagen Inverse Feature Visualization als einen ersten Schritt in Richtung automatisierter Verarbeitung von Merkmalsabbildungen vor, und zeigen dass unsere Methode der gradientenbasierten Inverse Feature Visualization zu einer gegebenen Merkmalsabbildung die zugehörigen Neuronen identifizieren kann.

Acknowledgments

First and foremost, I would like to thank Prof. Dr. Rüdiger Westermann. As my doctoral supervisor, he offered me the opportunity to work on a doctoral degree at the Technical University of Munich (TUM) and provided me with continuous support in all matters regarding administration, research, and publishing. Our frequent discussions were invaluable when developing research ideas, assessing results, or composing publications in the graphics and visualization community. I am impressed not only by the vast amount of input he can provide when brainstorming but also by his remarkable authoring skills when composing papers. Without his critical feedback and ability to find concise formulations, publications probably would have had twice the page count while being twice as difficult to follow along as a reader. Last but not least, I highly value his professional communication style not only in discussions with me but also in critical conversations with students. From my limited experience, only a small group of people has the ability to voice honest criticism without running into danger of getting even slightly personal. Prof. Dr. Rüdiger Westermann certainly belongs to this group. Even more, he always manages to turn setbacks into motivating suggestions on how to improve and keep on pursuing one's goals.

Beyond that, I would like to thank all my former and current colleagues for making each day in the office more than worthwhile. Thank you Alexander, Behdad, Bianca, Björn, Christoph, Fatemeh, Georg, Henrik, Josef, Junpeng, Kevin, Ludwig, Lukas, Marc, Marie-Lena, Mathias, Michael, Patrick, Philipp, Sebastian, Steffen, and many more for all the valuable discussions regarding both work and private matters, joyful moments, joint activities and welcome diversions when needed. More than two years of the pandemic have driven home the point that social interaction with all of you is something I would never like to miss out on again. Specifically, I am more than happy about having Sebastian as my office mate. He has been a great listener and advisor whenever I got stuck in my research. Sometimes, just telling a person about one's problems helps in finding the solution. Sebastian and I supported each other plenty in this regard, by taking on the role of what we call the (silent) "teddy bear" listener. Even more, we share common research interests with a slight bias towards mathematics,

as well as a passionate hobby of making and discussing music. In addition to that, I would like to thank Alexander as co-author on my very first publication at TUM. Collaborating with him in long nights to hit the deadline and then finally leaving the building at 2:00 am after submitting a paper is an experience I will never forget. Further, I thank Thorben Funke and Julia Liese for proofreading this doctoral thesis, Susanne Weitz for her excellent support in all administrative matters, and Sebastian Wohner for his relentless and quick support whenever I required a desktop environment for a student or managed to wreak havoc on the chair's GPU server or my desktop system.

Also, I would like to thank all persons outside of TUM who made my stage of life as a Ph.D. candidate special. Eternal gratitude goes to Julia Liese for her love, support, and plenty of endurance when coping with me in stressful times. The same holds for my ever-caring parents, who continue to share their experience of life with me and my brothers. Speaking of which, I would not be the same person without my two brothers who sparked my interests in mathematics and computer science early on. I thank both of them for being babysitters, idols, friends, and family all at once for their "little" brother. Similarly, I thank my nephews and my niece for their relentless source of energy, joy, and honesty. You are the best kids ever! Many thanks go to my various flatmates Amy, Hannah, Julia, Kevin, Linda, Pia, and Victoria over the course of the pandemic. In times of multiple lockdowns, they were the people who kept me sane. Likewise, I thank all my old friends from Hanover, who suddenly were as close as the neighbor next door. We had a lot of fun again in virtual board game rounds, chatting, and gaming. Finally, I wish to express my gratitude to all the warm-hearted and passionate members of the Sinfonic Wind Orchestra of Germering. Every rehearsal with them is a pure pleasure. Special thanks go to all fellow board members of the orchestra, for being friends and for dedicating loads and loads of time to keep the music going, without receiving a penny nor nearly as much recognition as would be deserved.

Contents

Abstract	v
Zusammenfassung	vii
Acknowledgments	xi
1 Introduction	1
1.1 Contributions	4
1.2 List of Publications	6
1.3 Outline	6
2 Related Work	9
2.1 Summed-Volume Tables	9
2.2 Voronoi Diagrams	12
2.3 Dimensionality Reduction	14
2.4 Neural Scene Representations	16
2.5 Level of Detail	20
2.6 Adaptive Super-Sampling	21
2.7 Explainable Artificial Intelligence	22
3 Parameterized Splitting of Summed Volume Tables	29
3.1 SVT Split Operation	31
3.2 Properties of the Data Structure	33
3.3 Heuristic for Building Parameter Trees	34
3.4 Addendum to the Paper	34
3.5 Alternative Representations of Summed-Volume Tables	35
4 Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques	39
4.1 Dimensionality Reduction	39
4.2 Voronoi Diagrams	42

4.3	Clustering	44
4.4	Matrix Seriation	47
4.5	PageRank	48
4.6	Robustness Plots	48
4.7	Representative Point Sets	50
5	Deep Learning	53
5.1	Training	54
5.2	Neural Network Architectures	56
6	Learning Generic Local Shape Properties for Adaptive Super-Sampling	61
6.1	Sparse Voxel Octrees & Level of Detail	62
6.2	PatchNet	63
6.3	Derivation of the Super-Sampling Pattern	66
7	Inverting the Feature Visualization Process for Feedforward Neural Networks	69
7.1	Inverse Feature Visualization	71
7.2	Gradient-Based Inverse Feature Visualization	72
8	Summary of Papers	77
A	Parameterized Splitting of Summed Volume Tables	77
B	Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques . .	79
C	Learning Generic Local Shape Properties for Adaptive Super-Sampling	80
D	Inverting the Feature Visualization Process for Feedforward Neural Networks	81
9	Conclusion	83
	Bibliography	87
	Published version of Paper A	105
	Published version of Paper B	123
	Published version of Paper C	137
	Version of Paper D as hosted on arXiv.org	141

Simulations of chaotic systems such as fluids and meteorological processes, as well as rendering applications in computer graphics, rely on accurate and thus highly resolved data representations in order to converge to a realistic output. Commonly, data is stored in spatial (3D) or spatiotemporal (4D) grids with resolutions of hundreds to thousands of samples per dimension, giving rise to giga- and terabyte scale datasets. In meteorological and climatological research, high-resolution historical weather data such as the publicly available ERA5 data set [Her+18; Her+20] containing global, atmospheric reanalysis data is often analyzed to indicate trends and reveal correlations between observed physical quantities. With a distance of 0.25° between samples in both latitude and longitude, 37 vertical pressure levels, and measurements every hour over more than forty years, the resulting data grid has a resolution of $1440 \times 721 \times 37 \times 350,000$, that is over 13T entries. On top of that, each entry stores a multivariate data sample of up to 16 different meteorological variables. When considering monthly averages, the dataset reduces to 18G entries.

Over the years, the notion of what is considered a large-scale dataset evolved with the constantly increasing memory capacities and compute capabilities of modern commodity hardware. Nowadays, generating and storing large-scale datasets of single-digit terabyte size is tractable on most mid-range priced desktop systems. However, the scalability of many applications is not defined by the generation process or storage requirements of the processed data, but by the scalability of the subsequent data processing algorithm. In general, with the ongoing growth of datasets, non-linear scalability becomes less practicable every year. For instance, a complexity of $\mathcal{O}(n \log n)$ introduced by many acceleration structures may already be too large to either fit the structure into memory or process it in reasonable time. For a 1TB dataset, the additional logarithmic factor accounts for a 40 times increase in complexity compared to linear algorithms. Hence, to fully utilize large-scale datasets, the design of scalable algorithms is paramount. To this end, we are going to present a specific data structure for

summed-volume tables with flexible control over its memory requirements, ranging from $\mathcal{O}(n \log n)$ down to $\mathcal{O}(n)$ at the cost of reduced access speed.

Besides high-resolution grid data, a second kind of large-scale datasets gained significantly in importance with the recent advances in data-driven designs, more specifically, deep learning (DL). Modern DL architectures are characterized by their sheer expressive power that stems from their highly parametrizable nature. Deep neural networks (DNNs) such as the well-known GoogLeNet [Sze+15] incorporate millions of adjustable parameters that are tweaked during a training process. Although the memory requirements for storing these parameters are quite manageable in the range of megabytes—networks have been used many times for compression schemes [DNJ20; Lu+21; Tak+21; WHW21]—the intermediate feature spaces arising when evaluating a deep neural network (DNN) are notoriously high-dimensional. In convolutional neural networks (CNNs) such as GoogLeNet, features are organized in multiple layers, with each layer outputting a multivariate 2D grid of features per input. Although the spatial resolution of each grid is limited, ranging from 7^2 to 112^2 for GoogLeNet, the number of features is comparably high, with up to 832 features being computed per 2D feature grid entry. In total, over 3.1M features are computed for a single input image when evaluating GoogLeNet. Even when considering small datasets with 60,000 images such as CIFAR10 [Kri09] (at the very moment ImageNet [Den+09] comprises more than 14 million images), this means that any technique that visualizes the feature space of a network over a prescribed dataset has to cope with ten thousands of a million-dimensional data points.

High dimensional data points as occurring in the context of DNNs pose significant challenges in subsequent reasoning, processing, and exploration. Much of our intuition about low-dimensional spaces does not translate to high dimensionality. Most striking, the Euclidean distance measure dictating the 3D world we perceive every day loses its significance in high-dimensional spaces, as randomly sampled points are all likely to have similar Euclidean distances to each other. In addition to that, the volume of spaces increases exponentially with the number of dimensions d and thus is difficult to conceptualize for $d \gg 3$. For the very same reasons, processing data in high-dimensional spaces is not trivial. The choice of distance measures has to be rethought, and reasonable sampling of high-dimensional spaces is not tractable due to the exponential volume growth. Last but not least, most existing visualization techniques are limited to encoding only a few dimensions by mapping them to glyph properties such as position, color, scale, and shape. Even visualization techniques that have been specifically designed to visualize multivariate data, such as parallel coordinate plots or heatmaps, are not applicable to datasets with millions of dimensions. Overall, high-dimensional data is not very accessible to both automated and manual inspection.

In practice, most high-dimensional dataset representations are overdetermined. Many combinations of values either cannot occur in a data generation process—due to constraints, dependencies, and correlations—or at least are very unlikely to occur, e.g. random pixel noise when taking a picture.

The intrinsic dimensionality of a specific dataset, that is the minimal number of variables required to represent the data, will be significantly smaller. Thus, a common conception of a high-dimensional dataset is that of a submanifold living in a high-dimensional ambient space. Data points arise as samples from the submanifold, with the dimension of the submanifold equalling the intrinsic dimension of the dataset. To increase accessibility, compact representations—i.e. the submanifold—of high-dimensional datasets have to be found. Dimensionality reduction (DR) serves this purpose by mapping data from a high-dimensional ambient space to a more benign, low-dimensional embedding space. However, popular DR techniques such as multidimensional scaling (MDS), t-distributed stochastic neighbour embedding (t-SNE) and uniform manifold approximation and projection (UMAP) produce different 2D point mappings for each run. Therefore, to obtain reliable insights into the data, multiple mappings have to be investigated and compared to each other. In this thesis, we are going to present a visualization technique that allows—at a glance—to identify regions for which most mappings coincide and thus can be considered reliable. Additionally, we present a data-driven scheme for finding low-dimensional representations of high-dimensional voxel scenes and apply it to support view-dependent, adaptive super-sampling in level of detail rendering.

Coming back to the initial example of feature spaces in GoogLeNet, some high-dimensional spaces may suffer from not only high dimensionality but also a lack of interpretability. As of yet, feature spaces in DNNs are not well understood, although they ultimately determine how a network comes to its conclusion. This is partially due to their high-dimensional nature, but also due to their complex relationships that form when concatenating various network layers. In order to facilitate a better understanding of feature spaces, DR techniques have been utilized to visualize them in interactive exploration tools [Kah+18; Rau+17; Car+19; Hoh+20]. In doing so, researchers were able to identify topological structures such as clusters and outliers regularly. Although this approach may be feasible to explain why a network fails to predict the correct outcome for a specific outlier, up to now neither it is exactly clear how such approaches translate into designing and training improved networks, nor do such visualizations reliably express the inter-feature relationships that are crucial in the network’s decision making process. In this regard, dimensionality reduction may even be detrimental. It rather obfuscates the meaning of axes in the embedding space, even if axes in the ambient space are associated with some interpretable concept. Instead, activation maximization (AM) [Erh+09; NYC19] was proposed as a means of associating visual and interpretable concepts with various features. Recently, AM has been used to define a taxonomy of feature classes by extensively investigating feature spaces of multiple networks [Cam+20]. In particular, resulting feature visualizations suggest that analogous features form across models and tasks [Ola+20]. However, these findings were only possible by manually comparing and matching a vast amount of feature visualization. To automate this process, we propose the approach of inverse feature visualization (IFV).

1.1. Contributions

In this thesis, four contributions specific to the issues we have just mentioned are presented. First, we propose a novel data structure for summed-volume tables (SVTs) that can flexibly trade access speed for lower memory consumption [RW21a]. In doing so, we ensure the scalability of SVTs even for large-scale datasets. Second, we address the inherent randomness of multiple DR techniques by providing a visualization that highlights stable regions over multiple runs of either t-SNE or MDS [RKW20]. Third, we present a novel encoder/decoder-based network architecture to learn low-dimensional representations over local geometry patches of large voxel scenes [RW22]. Last, we extend the idea of feature visualization by inverting the AM process [RW20]. That is, given a visualization of a feature created by AM, we can identify for which feature the visualization has been created. We envision that this technique will allow users to automatically detect similar features across multiple networks in an automated fashion. Note that the work addressing feature visualization has not been published in a peer-reviewed journal. In the following, the detailed contributions are given for each work of ours.

Parameterized Splitting of Summed Volume Tables [RW21a] We introduce a novel, hierarchical approach to computing and storing summed-volume tables (SVTs) by recursively splitting blocks of data before computing their prefix sums. Since partial prefix sums require fewer bits to encode their values, the overall memory consumption can be controlled by the number and position of the performed splitting operations. We implement a heuristic that automatically identifies close-to-optimal splits under any prescribed memory budget between $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$. In doing so, we provide a flexible SVT representation that can be tailored towards the user’s needs regarding access speed and/or memory consumption. We demonstrate the scalability of our approach by applying it to datasets of various sizes. In a comparative study, we compare our approach to various alternative SVT representations w.r.t. memory consumption, access speed, and qualitative features such as the complexity of read, update, and construction operations.

Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques [RKW20] We present a visualization technique to analyze the stability of neighborhood relations in an ensemble of 2D point embeddings, and we demonstrate its use for assessing the stability of dimensionality reduction techniques to variations in initialization. Although we present information over ensemble data, we intend to provide information “at a glance”, without making use of large dashboards or further brushing and linking over vast numbers of embeddings. Instead, we propose to use a novel stability measure for k -neighborhoods to identify clusters of embeddings via a matrix seriation technique, and then select a representative embedding by applying the PageRank algorithm. For each representative embedding determined in this way, we make use of k -order Voronoi diagrams

to analyze the stability of its local regions w.r.t. to the whole ensemble. A visual encoding of stable regions is provided by partitioning the 2D embedding, with the color of connected components indicating the stability of their corresponding regions and the color of separating ridges indicating the separation strength of adjacent components. We compare our approach against Member-centric cluster variability plots by Kumpf et al. [Kum+18] and show that both approaches can be used in conjunction to combine their strengths.

Learning Generic Local Shape Properties for Adaptive Super-Sampling [RW22] We propose a data-driven latent space encoding of local geometry patches that facilitates the identification of wrongly shaded pixels in level of detail (LOD)-based voxel rendering. It is realized by training a novel encoder/decoder-based network architecture, called PatchNet, that receives a local geometry patch as well as a viewport description, and then returns shading information about the geometry patch projected into the viewport. By comparing the shading information from the network to a 1-sample-per-pixel rendering, view-dependent discrepancies due to LOD approximations can be detected and corrected via adaptive super-sampling. Since we train on local geometry patches instead of whole scenes, our architecture generalizes to arbitrary scenes that are composed of similar geometry patches seen in training. Additionally, the viewport description is only available to the decoder, so that latent codes returned by the encoder are view-independent. This design enables the precomputation of all latent codes, which then are organized in a hierarchical LOD data structure to support coarse to fine look-ahead at various scales. We evaluate the super-sampling patterns returned by PatchNet against patterns of multiple other super-sampling strategies and show that PatchNet can detect shading errors that cannot be detected by any screen-space-based super-sampling strategy.

Inverting the Feature Visualization Process for Feedforward Neural Networks [RW20] We present the problem of inverse feature visualization (IFV). That is, given an arbitrary input, one intends to find a linear combination of neurons for which activation maximization will most likely converge to the given input. We solve this problem for specific inputs that have been obtained by activation maximization in the first place. Thus, we can recover the neurons for which activation maximization has been applied, by only having access to the feature visualization and its activations and gradients when fed to the network. Our method is realized by optimizing for those neurons for which the gradient of the activation maximization objective vanishes. Since this formulation allows for multiple trivial solutions (e.g. by dead neurons), several reformulations of the proposed optimization problem are performed to mask out trivial solutions and reliably recover the neurons of interest. In various experiments, it is shown that these reformulations are necessary and sufficient to invert feature visualizations. In discussions, we envision how IFV may be of use in deep learning

feature analysis and present directions on how to improve our algorithm such that it is applicable to any input instead of feature visualizations only.

1.2. List of Publications

Two of the four works accompanying the thesis have been published in full papers of the following peer-reviewed conferences and journals:

- Christian Reinbold and Rüdiger Westermann. “Parameterized Splitting of Summed Volume Tables”. In: *Computer Graphics Forum* 40.3 (2021), pp. 123–133. DOI: 10.1111/cgf.14294
- Christian Reinbold, Alexander Kumpf, and Rüdiger Westermann. “Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques”. In: *Computer Graphics Forum* 39.1 (2020), pp. 333–346. DOI: 10.1111/cgf.13806

One work of the thesis has been published in a *short paper* of the peer-reviewed conference Eurographics 2022. As a short paper, it is not assessment-relevant according to the TUM Regulations for the Award of Doctoral Degrees, effective January 1, 2014:

- Christian Reinbold and Rüdiger Westermann. “Learning Generic Local Shape Properties for Adaptive Super-Sampling”. In: *Eurographics – Short Papers*. Ed. by Nuria Pelechano and David Vanderhaeghe. The Eurographics Association, 2022. ISBN: 978-3-03868-169-4. DOI: 10.2312/egs.20221032

Lastly, one work is available on arxiv.org. It has *not been published* in a peer-reviewed process and thus is not assessment-relevant according to the TUM Regulations for the Award of Doctoral Degrees, effective January 1, 2014:

- Christian Reinbold and Rüdiger Westermann. “Inverting the Feature Visualization Process for Feedforward Neural Networks”. In: *arXiv e-prints* (July 2020). arXiv: 2007.10757 [cs.LG]

1.3. Outline

The thesis is structured as follows. In Chap. 2, prior work related to our publications is discussed. Chap. 3 to 7 contain the concepts and approaches that are either used or introduced in the works associated with this thesis. Our novel SVT data structure [RW21a] is described in Chap. 3, and the visualization technique to analyze the stability of 2D point sets [RKW20] is discussed in Chap. 4. To prepare Chap. 6 and 7, a compact introduction to fundamentals of deep learning is presented in 5. Then, Chap. 6 addresses learning generic local shape properties for adaptive super-sampling [RW22].

Inverse feature visualization [RW22] is discussed in Chap. 7. After presenting summary information of our publications in Chap. 8, we conclude the thesis and present future research directions in Chap. 9.

In this chapter, we are going to discuss prior work that is related to this thesis. With respect to our work about summed-volume table representations [RW21a], we present the foundations of summed-volume tables, alternative representations, applications and efficient algorithms for computing a summed-volume table. Further, we address applications of Voronoi diagrams, specifically in visualization, as well as dimensionality reduction techniques. Both topics are related to our work about visualizing stable regions [RKW20]. In the context of learning generic local shape properties for adaptive super-sampling [RW22], we discuss neural scene representations, level of detail applications, and adaptive sampling techniques. Lastly, we present recent developments in the field of explainable artificial intelligence which we consider to be important for inverse feature visualization [RW20]. Here, we also emphasize robust training as a technique that facilitates the forming of human interpretable features.

2.1. Summed-Volume Tables

Summed-area tables (SATs) are a versatile data structure which has initially been introduced to enable high-quality mipmapping [Cro84] by precomputing integral values over rectangular regions in a 2D data array F . They are realized by populating an array of equal size as F such that the value at index (x, y) equals the sum over all values contained in the rectangular subarray that is spanned by the origin and (x, y) . With four values from a SAT, the integral over any rectangular region can be obtained in constant time. The concept of SATs is applicable to arrays of arbitrary dimension d [Tap11], in which case the data structure is termed a d -dimensional summed-volume table (SVT). As shown in Chap. 3, SVTs allow to compute integrals over any hyperbox by querying 2^d values. In other words, integrals can be computed in $\mathcal{O}(1)$ independently of the hyperbox or array size. However, SVTs have two significant limitations, which we discuss in detail in Sec. 3. First, they have

a large memory footprint of $\mathcal{O}(n \log n)$, where n is the number of elements in F . The additional logarithmic factor is especially costly for large 3D or even higher-dimensional datasets. Second, the SVT-accelerated evaluation of integrals suffers from catastrophic cancellation when performed with floating-point arithmetic [Hen+05].

Summed-Volume Table Representations Several SVT representations have been proposed to address the aforementioned limitations. Note that although some authors discuss their modifications in the context of two-dimensional SATs, all concepts presented here also generalize to SVTs of arbitrary dimensionality. Hensley et al. [Hen+05] suggest to mitigate precision loss issues for floating-point SATs by offsetting data values before computing SAT entries. Further, they suggest computing separate SATs for the four quadrants of the image. Both measures come at no additional costs during integral computation, and they decrease the dynamic range of floating-point values stored in a SAT so that the impact of cancellation effects is reduced. The suggestions of Hensley et al. likewise can be used to reduce precision requirements and thus the memory footprint of integral SATs. From now on, all related work presented here assumes integral SVTs. As stated in Sec. 3, this prerequisite is reasonable and can always be established by quantization.

Belt [Bel08] proposes to apply rounding by value truncation to the input array before computing its corresponding SVT. By reducing the precision of the input, the SVT requires fewer bits of precision as well. To compensate for rounding error accumulation, the rounding routine is improved by considering introduced rounding errors of neighboring SVT entries. Note that this technique can also be utilized when quantizing floating-point arrays to prepare them for a SVT representation expecting integral inputs. In the same work, Belt also introduces the technique of computation through the overflow. Here, SVT entries are stored modulo 2^ℓ by dropping all except for ℓ least significant bits. As a consequence, memory requirements reduce to $\mathcal{O}(\ell n)$. As long as integrated hyperboxes are small enough such that results greater or equal than 2^ℓ are omitted, all computations can be performed modulo 2^ℓ to obtain correct results. Computation through the overflow excels in filtering tasks with kernels of prescribed box size. However, it is not applicable in situations where box sizes are either large or not known in advance. Further representations of integral SVTs have been proposed in works of Ehsan et al. [Ehs+15], Zellmann et al. [ZSL18], Urschler et al. [UBD13] and Schneider and Rautek [SR17]. All of these methods have in common that they reduce the amount of required memory at the cost of reading more values from the SVT representation when computing an integral. Concise explanations of each method are given in Sec. 3.5.

Efficient Generation of Summed-Area Tables SATs are generated in linear time by performing a one prefix sum scan along the rows, and a second one along the columns to sum up row-wise prefix sums. Bilgic et al. [BHM10] implement the prefix sum scan along the rows on the graphics

processing unit (GPU) where prefix sum scans along contiguous elements in memory can be efficiently realized. Then, the array of row-wise prefix sums is transposed by a block-wise operating kernel that exhibits a coalescent memory access pattern and avoids bank conflicts by additional padding. Due to the transposition, columns in the original data array are arranged in contiguous memory now, so that the second prefix sum scan along columns is computed efficiently as well. Overall, this approach requires three read-write operations in global memory per array entry. The block-register-local-transpose-scanrow (BRLT-ScanRow) method as proposed by Chen et al. [Che+18] performs the row-wise prefix sum scan and subsequent transpose operation jointly in shared memory before writing the results to global memory. By invoking this routine twice, a SAT can be computed with only two read-write operations in global memory. Emoto et al. [Emo+18] improve further on this by presenting an algorithm that requires approximately one read-write operation in global memory per array entry. As a consequence, they achieve a runtime performance that is 5.7% above the theoretical lower bound given by the runtime performance of matrix duplication.

Applications SVTs have numerous applications in graphics, object detection, and region filtering as they allow the evaluation of arbitrarily sized box filters in $\mathcal{O}(1)$. In graphics, Hensley et al. [Hen+05] apply SATs to realize glossy reflectors and refractors. Whenever a surface is sampled, the reflected (or refracted) object’s texture map is filtered with a box filter of a kernel size depending on the distance between the reflector and the reflected object. Similarly, depth-of-field effects are implemented by a post-processing step that applies box filters of various sizes to the rendered image. SATs were also used to realize soft shadow effects, by adaptively blurring variance shadow maps with various filter sizes depending on the distance to the light blocker [Lau07].

SATs have been successfully used to accelerate classical object detection algorithms that base upon box filtering. In 2004, Viola et al. [VJ04] utilize SATs in a method to detect faces by convolving the input image with a simple set of piecewise box filter kernels. Grabner et al. [GGB06] propose to speed up the scale invariant feature transform (SIFT) method [Low04] that is used to generate feature descriptors for object detection. SIFT detects scale-invariant points of interest by applying box filters of various sizes to the image. Similarly, the computation of keypoint descriptors in SIFT can be sped up by realizing integration over 4×4 image gradient patches with SATs. Bay et al. [BTV06] propose a similar method for keypoint detection. Hessian matrices at various scales are approximated by convolving with piecewise box filter kernels via SATs. Then, local maxima of the Hessian determinant in 3D scale space are identified as points of interest. In 2014, Facciolo et al. [FLM14] deploy SATs to speed up block matching, that is finding similar patches between two images with close by centers. For each eligible center offset, all possible block pairs with the given center offset can be compared efficiently by building a SAT over a difference image that is obtained after shifting one of the two input images according to the offset. Porikli [Por05] implements a SAT over histograms to accelerate

the search for image subregions with similar color histograms as the object of interest. Also, SATs can speed up an adaptive image binarization method with applications in classical optical character recognition [SKB08].

In region filtering, SATs can be utilized when approximating arbitrary filters with box filters [BSB10]. Hussein et al. [HPD08] factorize arbitrary filter kernels into a linear combination of region-independent functions h_i and a point-independent functions g_i . Whereas each g_i is evaluated in $\mathcal{O}(1)$ when filtering a region, the functions h_i have to be integrated, which can be realized in $\mathcal{O}(1)$ by precomputing a SAT for each region-independent function. Phan et al. [Pha+12] show that SATs enable the computation of statistical quantities for arbitrarily large axis-aligned regions in constant time, and use this insight to speed up an existing image quality assessment algorithm.

Lastly, SVTs allow to test regions for emptiness in $\mathcal{O}(1)$ by evaluating an integral over an occupancy mask and comparing it to zero. In this context, SVTs have been used in the creation of acceleration structures for sparse scenes and data. Havran et al. [HHS06] build a bounding volume hierarchy (BVH) / spatial k-d tree hybrid acceleration structure for mesh data by discretizing the 3D domain and finding kd-splits in expected $\mathcal{O}(\log \log n)$. A SVT over the discretized domain is then used to evaluate the split cost function in constant time. Vidal et al. [VMD08] propose to use SVTs to speed up cost function evaluations in a BVH construction process for voxelized volume datasets. In their work, the cost function requires the computation of bounding volumes over binary occupancy data. By running binary search on a SVT, this task can be solved in $\mathcal{O}(\log n)$ instead of $\mathcal{O}(n^3)$, where n is the side length of the volume. Ganter and Manzke [GM19] propose to use SVTs to cluster bounding volumes of small size before assembling them bottom-up into a BVH for direct volume rendering. Schneider et al. [SR17] make use of 1D SVTs to realize a spatial indexing structure into sparse volumes.

2.2. Voronoi Diagrams

Voronoi diagrams as introduced by the eponymous author in 1908 [Vor08] have been proven to be a ubiquitous data structure, with a wide variety of generalizations and applications in numerous fields [Oka+00]. The classical Voronoi diagram partitions the ambient space of a finite point set P of Voronoi sites into a set of disjoint regions, called Voronoi cells. Each cell is defined by all locations in the ambient space that share the same nearest neighbor in P . In particular, Voronoi cells and points in P are in one-to-one correspondence. By replacing the distance measure according to which nearest neighbors are defined, various generalizations of Voronoi diagrams such as the power diagram [Aur87] can be obtained. Here, the Euclidean distance measure is replaced by the square distance minus the square of a point-specific weight $w(p)$. That is, the distance between a location x in the ambient space and a point $p \in P$ is given by $d(x, p) = \|x - p\|^2 - w(p)^2$. By leaving out

the squares in the distance formulation, additively weighted Voronoi diagrams are obtained. Similarly, multiplicatively weighted Voronoi diagrams are defined by $d(x, p) = w(p) \cdot \|x - p\|$. In practice, weights can be utilized to adjust the size of Voronoi cells according to application-specific constraints. Centroidal Voronoi diagrams [DFG99] are defined by the constraint that each point has to be the centroid of its respective Voronoi cell. They can be created by Lloyd’s algorithm [Llo82].

Another generalization of Voronoi diagrams is realized by not requiring Voronoi sites to be points anymore. Instead, a Voronoi site can be an arbitrary geometrical shape, or more generally, a (not necessarily finite) subset of locations in the ambient space. In this case, the distance measure between a site p and location x is given by $d(x, p) = \min_{y \in p} \|x - y\|$. Last but not least, k -order Voronoi diagrams [SH75] generalize regular Voronoi diagrams by organizing locations into regions for which the k -nearest neighbors coincide. In our work [RKW20], we utilize the topology of k -order Voronoi diagrams (see Sec. 4.2) to cluster k -neighborhoods. In graphics, Voronoi diagrams are used in procedural mesh generation and surface reconstruction. For instance, Olsen utilizes Voronoi diagrams to break monotony in noise-based terrain generation [Ols04]. Bletterer et al. [BPA22] construct surfaces from depth maps by first computing a localized, centroidal Voronoi tessellation per depth map, and then gluing them to a global Voronoi tessellation representing the 3D surface.

Applications in Visualization Voronoi diagrams exhibit several properties of particular interest for the visualization community. First, Voronoi tessellations do not suffer from overplotting as Voronoi cells do not overlap. Second, clutter is avoided as long as values mapped to visual attributes of Voronoi cells have spatial continuity. Then, even thousands of Voronoi cells appear as large, homogeneous regions. Third, Voronoi tessellations can be rendered in the background of visualizations that leave large portions of the screen empty otherwise. For instance, we make use of this property when visualizing glyph representations of data points along with robustness plots [RKW20]. Last, the one-to-one correspondence between Voronoi cells and points is easy to perceive since a point is always located in its respective Voronoi cell. Note however that this property is lost for higher-order Voronoi diagrams. Thus, Aupetit [Aup07] proposes to visualize quantities over pairs of points by Segment-Voronoi diagrams instead of 2-order Voronoi diagrams.

Balzer and Deussen propose Voronoi treemaps [BD05] to layout treemaps with more flexible, non-rectangular shapes. To lay out the children of a tree node, a Voronoi tessellation is created by sampling a point per child from the area assigned to the current node. In an iterative process, each point is processed by a) updating its weight according to the actual and expected size of its Voronoi cell, and b) moving it to the centroid of its Voronoi cell. The areas assigned to the children are given by the Voronoi cells after the iterative process converges to a stable, centroidal Voronoi tessellation. Nocke et al. [NSB04] utilize Voronoi tessellations to generate a space-filling visualization depicting aggregated values of climate data clusters. Aupetit [Aup07] visualizes a proximity measure assessing

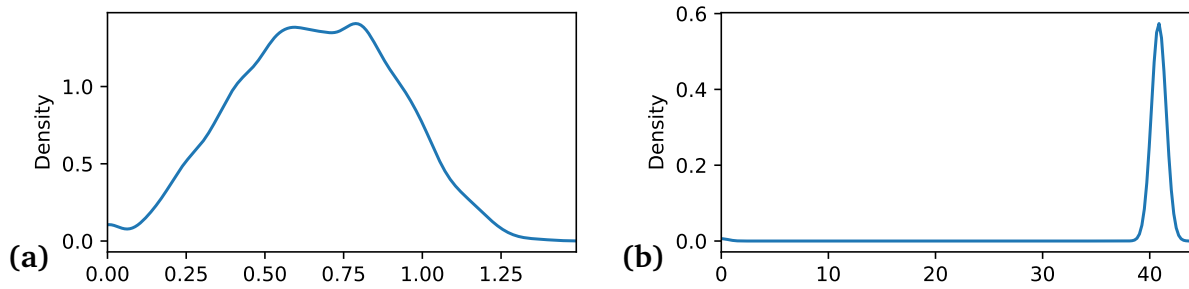


Figure 2.1.: Distribution of pairwise Euclidean distances for 100 points sampled from a (a) 3D and (b) 10,000D unit cube, respectively.

the quality of dimensionality reduction (DR) projections using colored Voronoi cells. Later, Lespinats and Aupetit [LA11] extend on this idea and map a two-dimensional color-code to Voronoi cells for the visualization of distortions such as tears and false neighborhoods introduced by DR techniques.

2.3. Dimensionality Reduction

Processing, exploring, or visualizing high-dimensional data is challenging not only due to the increased computational efforts required to process the data but also due to our lack of intuition when reasoning about more than three-dimensional data. For instance, the significance of Euclidean measures declines in high-dimensional spaces, as most pairs of points are likely to have similar Euclidean distances, see Fig. 2.1. Additionally, since the volume of spaces increases exponentially with the number d of dimensions, dense sampling of high-dimensional spaces is not tractable. On the visualization side, most visualization techniques cannot cope with high dimensional data as the number of visual attributes that can be used for mapping dimensions is limited. To bypass issues arising from high dimensionality, a plethora of DR techniques have been developed to map data from a high-dimensional ambient space to a more benign, low-dimensional embedding space. More than 70 approaches are mentioned in the survey of Espadoto et al. [Esp+21, Table 1]. Here, we can only discuss the most popular DR techniques and refer the interested reader to the work of Espadoto et al. as well as the surveys mentioned there for an extensive overview of the field.

Feature selection [Li+17] is the most straightforward approach to DR by identifying a subset of dimensions that contain the relevant information of a dataset. The remaining dimensions with no— or redundant—information are removed to obtain a tractable dataset. When including an additional dimension decorrelation step before applying feature selection, one arrives at the principal component analysis (PCA) [Pea01]. It computes the directions of the dataset showing the most variance by decomposing its covariance matrix. A reduction to a k -dimensional embedding space is realized

by projecting all data points onto the subspace spanned by the k directions with the most data variance. Since the projection from ambient to embedding space can be described by a linear transformation, PCA is considered a linear method. However, it is not guaranteed that high-dimensional datasets can be solely explained by a limited set of linear relationships. Data points may be located on highly non-linear submanifolds embedded in the high-dimensional ambient space. Non-linear DR techniques intend to identify these submanifolds and use their presumably low intrinsic dimensionality to project the dataset to the low-dimensional embedding space. t-distributed stochastic neighbour embedding (t-SNE) [MH08] models the neighborhoods of data points in ambient and embedding space separately as two discrete probability distributions. After randomly initializing points in embedding space, their locations are updated in an iterative optimization scheme minimizing the Kullback-Leibler divergence between the ambient and embedding probability distribution. By considering neighborhood relationships, manifold structures are locally preserved in the embedding space. The original t-SNE formulation has a runtime complexity of $\mathcal{O}(n^2)$. Van der Maaten [Maa14] presents an approximation of t-SNE that can be obtained in $\mathcal{O}(n \log n)$. In the embedding space, distances can be approximated by a technique first introduced by Barnes and Hut [BH86]. Points are grouped in hyperoctree cells such that similar distances to all points in a cell of sufficiently small size can be assumed. In ambient space, neighborhood probabilities are thinned out by considering only a fixed number of nearest neighbors, which can be efficiently derived from a vantage-point tree [Yia93].

Uniform manifold approximation and projection (UMAP) is a more recent DR technique that is based upon matching fuzzy topologies of edges in the ambient and embedding space. Each edge appearing in the k -nearest neighborhood graphs over point locations in both ambient and embedding space is assigned a probability value depending on the distance of connected points as well as the distance of both points to their respective nearest neighbor. Then, the point positions in embedding space are optimized such that the cross-entropy between fuzzy edges in ambient and embedding space is minimized. Compared to t-SNE, probability relations in embedding space are sparse, so no further hyperoctree is required as acceleration structure. In particular, UMAP scales better than t-SNE concerning dimensions of both the ambient and embedding space, making reductions to embedding spaces with more than two dimensions feasible.

Whereas t-SNE and UMAP optimize for local connectivity of points, multidimensional scaling (MDS) techniques are global in the sense that they intend to match distances between any pairs of points. This is achieved by minimizing a stress function over the embedded point positions that becomes zero if and only if distances are preserved perfectly. As discussed in surveys by Gronen and van de Velden [GV16], as well as Saeed et al. [Sae+18], various stress functions and algorithms to minimize them have been proposed over time. Here, we highlight two specific MDS techniques and refer to the surveys for others. Scaling by majorizing a complicated function (SMACOF) [LM09] by de Leeuw and Mair solves a least-squares MDS stress function by iteratively optimizing and updating a convex

majorization of the stress function (see Sec. 4.1 for details). Isomap [TdL00], on the other hand, is an MDS variant where Euclidean distances in embedding space are matched against geodetic distances in ambient space. Geodetic distances are obtained by computing the weighted k -nearest neighborhood graph in ambient space and then using a shortest path algorithm on this graph to determine geodetic distances. Similar to t-SNE and UMAP, this approach intends to recover the submanifolds the data is located in. However, by considering points with far geodetic distances as well, Isomap attempts to match the global structures of a submanifold instead of local ones.

Quality Assessment In general, the high-dimensional ambient space cannot be projected to the low-dimensional embedding space without loss of information. Whereas linear methods such as PCA explicitly squash dimensions, non-linear methods introduce distortions when mapping high-dimensional points to low-dimensional ones. For 2D embeddings, Aupetit [Aup07] proposes to visualize the local quality of a projection by picking a single point and mapping all distances to other points in high-dimensional space to a color of the respective other point’s Voronoi cell in embedding space. Thus, distortions become visible by discontinuities and non-circular shapes in the resulting coloring of a Voronoi diagram tessellating the embedding space. They also propose to use Segment-Voronoi tessellations to highlight false neighborhoods in the projection. Hermann et al. [HGK09] visualize local Kruskal stress by mapping it to point color in a scatter plot. It measures distortions in distances between a data point and its closest neighbors in the projection. Over the years, various quality assessment metrics have been introduced to evaluate the credibility of projections. We refer to the work of Venna [Ven07, Section 3] for a survey of metrics that measure and visualize the quality of projections.

2.4. Neural Scene Representations

In classical computer graphics, geometric scenes and scientific datasets have been described by polygonal surface meshes, point clouds, implicit functional descriptions, or voxel models. Due to the discrete nature of most representations, they lack differentiability and cannot easily be integrated into existing deep learning pipelines. Recently, various differentiable scene representation networks (SRNs) architectures have been developed to encode geometric and volumetric data in 3D space. They allow for the integration of geometric models into training pipelines so that various geometric tasks in computer vision and graphics become amenable to deep learning frameworks. SRNs have successfully been used in fields such as shape completion, shape reconstruction, scene compression, adaptive sampling, super-resolution, and global illumination.

Most neural scene representations are realized by learning either an implicit function description $\mathbb{R}^3 \rightarrow \mathbb{R}$ or a volumetric, explicit function description that maps from \mathbb{R}^3 to volumetric properties such

as material density or color. Whereas the former approach is suitable for modeling and rendering iso-surfaces, the latter approach is closer to direct volume rendering (DVR)—although it also has been used for modeling surfaces with sparse, volumetric density fields [Mil+20]. Besides being differentiable, SRNs have several other benefits compared to classical approaches. First, they are continuous by design and do not depend on a specific spatial resolution. The quality of SRNs rather depends on the number of weights that are available to approximate a geometric shape [DNJ20]. In particular, a SRN can adaptively allocate more weights to regions with fine geometric details. Second, SRNs can incorporate geometric priors from a training dataset, so that the space of representable surfaces is aligned with physically plausible surfaces [CAP20]. This is not true for classical scene representations, where random initialization most certainly leads to highly fragmented, implausible scenes. In particular, this prior allows SRNs to store scenes far more compactly than classical approaches [DNJ20]. On the downside, accessing and rendering neural scene representations is comparably slow as of yet, since each sample of the scene has to be computed on the fly by a costly invocation of the SRN.

Implicit Representations DeepSDF by Park et al. [Par+19] has been one of the first implicit SRNs, showing that they are capable of learning geometric shapes as well as strong geometric priors for shape completion. DeepSDF trains jointly on a collection of shapes by learning an implicit function $f: \mathbb{R}^3 \times Z \rightarrow \mathbb{R}$, where the additional latent code input $z \in Z$ selects the respective shape in the collection. That is, the shape associated with z is given by the implicit function $x \mapsto f(x, z)$. This design enables smooth shape interpolation as well as shape completion from a limited set of signed distance function samples. Niemeyer et al. [Nie+20] show that rendering implicit surfaces can be made differentiable through implicit differentiation. Hence, SRNs can now be trained on multiple 2D views of an object to capture its view-consistent 3D shape in an implicit SRN and texture in another SRN mapping position to color. Similar to DeepSDF, both networks can be conditioned with a geometric prior by adding a latent code z to the input. As a consequence, even single view object reconstruction is possible by hallucinating occluded geometry according to the prior. Yariv et al. [Yar+20] extend the approach of Niemeyer by a) additionally optimizing for camera parameters to compensate for inaccurate camera poses in multi-view datasets and b) replacing the texture network with a more sophisticated shading network that does not only receive a 3D surface-ray intersection point, but also surface normal, viewing direction, and a scene-specific global geometry feature vector.

Chibane et al. [CAP20] propose an encoder/decoder structure for shape completion. A 3D convolutional neural network (CNN) encodes the incomplete input shape at various levels of detail into latent feature grids. The decoder implements an implicit function describing the completed shape. It is realized by sampling the multi-scale latent feature grids at the input position and translating the sampled feature vectors to the implicit function value. Regarding compression, the work of Davies et al. [DNJ20] demonstrates that SRNs can be used for lossy compression of geometry,

with significantly higher compression rates than classical mesh decimation techniques. Similarly, Takikawa et al. [Tak+21] achieve impressive compression rates by masking out empty space with a sparse voxel octree (SVO) and encoding geometry in a level of detail hierarchy of implicit SRNs.

As can be seen from the previous paragraphs, plenty of implicit SRNs have been introduced over the years, solving problems in various research fields. Here, we only have presented the papers that did draw our attention when working on our topic of learning generic local shape properties [RW22]. For further research, we refer the reader to an extensive list of works curated by Sitzmann [Sit20].

Volumetric Representations In contrast to implicit representations, volumetric SRNs intend to learn the volumetric properties of the scene directly. They became popular with the work of Mildenhall et al. [Mil+20] presenting the neural radiance field (NeRF). Their work borrows from techniques of DVR to approximate surfaces in 3D space. By sampling a SRN that transforms 3D positions and view directions to color and density values along view rays, a volumetric, view-consistent 3D representation of surface models can be learned from multiple views. Many follow-up works have been inspired by NeRF [Lin20]. For instance, Lie et al. [Liu+20b] present a NeRF variant operating over sparse voxel grids. The 3D positional input to NeRF is replaced by a latent feature sampled from the voxel surrounding the current 3D position. Then during training, latent voxel features are trained to encode the voxel’s geometry. Further, the voxel grid is progressively refined and thinned out by frequently doubling the voxel grid resolution and removing all voxels for which the neural scene representation network only returns volume densities close to zero. Compared to NeRF, rendering performance is improved by sampling the scene representation network only within allocated voxels. Hedman et al. [Hed+21] reformulate the NeRF architecture to enable real-time rendering. They propose to split NeRF into a view-independent and view-dependent part. After training, view-independent latent features, volume densities, and diffuse colors are baked in a sparse voxel grid representation. During rendering, all view-independent quantities are sampled directly from the voxel grid. Only view-dependent specular lighting is derived from a small scale multilayer perceptron (MLP) that requires the integrated volume density, the integrated latent feature, and the current viewing direction as input. Garbin et al. [Gar+21] split NeRF into an SRN encoding positional information in latent features over a 3D domain, and an SRN encoding view-dependent information in latent features over a spherical domain. The position- and view-dependent object color is obtained by combining the output of both networks via the dot product. After factorizing the 5D input domain of NeRF into independent 3D and 2D domains, outputs can be fully baked by sampling both networks along high-resolution 3D and 2D grids and storing non-zero outputs in a sparse data structure.

NeRF makes explicit use of the DVR pipeline by integrating density and color along view rays. DeepVoxels by Sitzmann et al. [Sit+18], on the other hand, follow a different approach involving 2D CNNs. Given a set of renderings of an object, feature maps are extracted by a 2D convolutional

encoder and—based on current camera parameters—lifted to a 3D feature volume encoding the actual geometry. To render a new view, the 3D feature volume is projected to the canonical view volume, where features are blended with an occlusion network. Finally, the 2D image of blended features is converted to an RGB image by applying a 2D convolutional neural renderer. In a later work [SZW19], the architecture is refined by a) replacing the 3D feature volume with a scene representation network mapping 3D positions to latent features, b) dropping the 2D→3D uplifting and 3D→2D projection module in favor of a ray marching long short-term memory module, and c) changing the final neural rendering network architecture from a CNN to a per-pixel MLP.

Similar to implicit SRNs, volumetric SRNs can be utilized to realize lossy compressions over volumetric datasets [Lu+21]. By storing network weights instead of grid-based data representations, significant memory savings can be achieved at the cost of an expensive forward pass through the network when decoding a value. To further increase compression performance, an additional, coarse grid of latent features can be learned that provides the SRNs with further geometry cues. Weiss et al. [WHW21] suggest a similar approach to compress ensembles of 1D density and 4D color volumes by small-scale MLPs. In particular, they provide real-time decoding performance by utilizing modern Tensor Cores and keeping all activations in shared GPU memory when evaluating the MLP. Müller et al. [Mül+22] propose a similar approach to Weiss et al. with real-time performance to represent gigapixel images, scene geometry, and global illumination effects. However, they realize the memory backend of the latent feature grid via a hash table with no explicit collision handling. Instead, they rely on the network to resolve collisions. Thus, sparse latent feature grids with exceptionally high virtual resolutions can be realized.

Local Geometry Encodings All SRNs presented so far learn the whole scene geometry. A couple of works focus on learning encodings of local geometry patches instead. Jiang et al. [Jia+20] suggest encoding scenes by transforming local geometry patches to latent feature codes, which then act as a neural scene representation. The actual scene geometry can be recovered by evaluating a decoder network that converts latent features to implicit function samples. Thus, any surface encoded by latent feature codes can be extracted with isosurface techniques such as marching cubes, ray marching, or sphere tracing. The respective encoder/decoder architecture is trained once on a sufficiently complex scene, and then can be used to describe arbitrary scenes that are composed of similar geometry patches seen in training. In particular, both en- and decoder can be used for new scenes without any costly retraining. The authors use the latent feature space as a strong geometric prior to extract implicit surfaces from point clouds. This is achieved by optimizing over latent patch features such that the implicit function of the scene evaluates to zero at points in the point cloud. Liu et al. [Liu+20a] propose to encode local geometry information to realize a data-driven extension of Loop Subdivision where vertex positions are updated based on a stack of neural networks. A first network encodes the

1-ring neighborhood of each vertex in a latent feature vector, which is subsequently processed by a second and third network to update positions of old and newly introduced vertices, respectively. Chen et al. [Che+21] propose to upsample voxel geometry by training a generative adversarial network (GAN) to encode details of local geometry patches into style codes. Depending on the input style code, specific fine geometric details are fused onto arbitrary coarse voxel geometries by the GAN encoder.

2.5. Level of Detail

Level of detail (LOD) techniques [Lue+03] in computer graphics create a multi-resolution hierarchy of filtered representation of one and the same entity. Depending on the frequency at which an entity is sampled by a viewport, one representation is chosen and rendered. LOD techniques do not only increase rendering performance by reducing the complexity of distant entities but also avoid sampling below the Nyquist rate by limiting the bandwidth of the original entity. In doing so, aliasing artifacts and temporal noise during camera movement are avoided.

Textures Mipmapping [Wil83] creates a LOD hierarchy of textures by recursively downscaling them by a factor of two until only one texel remains. A novel LOD technique for uv-free texture mapping is proposed by Dolonius et al. [DSA20]. They propose to implement textures by storing surface texture values in an SVO that is aligned with the surface shape. A LOD hierarchy is realized by storing value averages at each internal node.

Surface Geometry For 3D geometry, LOD hierarchies can be created by mesh decimation techniques such as progressive meshes by Hoppe [Hop96]. If a mesh decimation technique supports continuous collapse and expansion operations (i.e. geomorphing), popping artifacts at level transitions can be avoided by interpolating the respective operations that transform the geometry to the next level representation. A novel, deep learning (DL)-based technique by Takikawa et al. [Tak+21] implements a continuous LOD representation of implicit surfaces by encoding a signed distance function (SDF) into latent features organized in an SVO. To sample the SDF, latent features are sampled from all SVO levels, summed, and then processed by a neural decoder outputting the signed distance.

Point Clouds Layered point clouds by Gobbetti and Marton [GM04] create a LOD hierarchy of point clouds in a top-down process by partitioning all points over multiple levels of a binary space partition tree. Each node stores at most M uniformly distributed points, so that rendering the point cloud with a prescribed point density is achieved by rendering only nodes up to a fixed depth depending on M . Fraedrich et al. [FSW09] propose a bottom-up process for building a LOD octree hierarchy of

particles with varying radii. At each level, particles are thinned out according to two strategies. First, all particles with small radii such that they project to a single pixel w.r.t. the current LOD are merged to a single particle of zero radius, i.e. a point. Second, particles with similar radii are merged as soon as the difference in radius drops below a threshold that is multiplicative w.r.t. to the voxel size at the current octree level.

Terrain Schneider and Westermann [SW06] implement multi-level digital elevation models (DEMs) by adaptively refining restricted quadtree meshes until a level-dependent upper error bound is satisfied. During rendering, screen-space gradients determine at which LOD a specific tile of the terrain is sampled. Dick et al. [DSW09] propose a compression scheme for restricted quadtree mesh hierarchies that allows to efficiently stream large DEMs into GPU memory. Progressive mesh refinements are encoded by paths along triangle strips, where each path element can be derived from the previous one by distinguishing up to six cases. Efficient decoding of paths to triangle strips is realized by splitting paths into a large number of subpaths, that then are decoded in parallel by the GPU.

2.6. Adaptive Super-Sampling

As discussed in Sec. 2.5, sampling entities below the Nyquist rate results in noticeable sampling artifacts such as aliasing and temporal noise. Level of detail techniques ensure a sufficient sampling rate by limiting the frequency of the sampled signal. Vice versa, one can also increase the sampling frequency instead, which is called super-sampling. In general, super-sampling is difficult to achieve due to hardware limitations and real-time constraints. Adaptive super-sampling serves as a compromise by super-sampling only the part of a signal with critical, high-frequency information. For instance, Mitchell [Mit87] proposes to refine sampling patterns at locations where the contrast between neighboring samples exceeds a user-defined threshold. Weiss et al. [Wei+20] train a network that derives an adaptive sampling pattern from a low-resolution rendering such that the high-resolution rendering can be reconstructed from only a few samples. The output image is created by inpainting the samples via a pull-push algorithm, and then correcting the inpainted result with a convolutional reconstruction network.

Adaptive sampling techniques are especially important in Monte Carlo ray tracing where stochastic light path sampling induces high variance in the estimated pixel values. Thus, low sample-count renderings contain a significant amount of visually distracting noise which gradually declines when adding more samples per pixel. However, Monte Carlo renderings of complex scenes may require thousands of samples per pixel to converge to a noise-free image. Generating so many samples is not feasible on most commodity hardware, and will certainly not be possible in real-time. Instead

of drawing a fixed number of samples per pixel, adaptive super-sampling stops if the already drawn samples satisfy a notion of stability, i.e. low variance.

A straightforward stopping criterion is realized by assuming a normal sampling distribution and stopping sampling as soon as the difference between the current and the true sample mean passes a confidence interval test [LRU85]. Rigau et al. [RFS03a] propose an entropy-based stopping criterion by reinterpreting luminance sample values for a single pixel as probabilities in a discrete probability distribution P . If the Shannon entropy of P is sufficiently close to the maximal attainable entropy (it would be reached if all probabilities and thus sample values are equal), sampling is stopped. This approach has been modified in several ways. One can derive probabilities from geometric properties such as surface normals and depth at the ray-geometry intersection point [RFS03a]. Further, it has been proposed to replace the Shannon entropy measure with Tsallis entropy [Xu+07]. Instead of measuring entropy, one can also measure the distance of P to the uniform probability distribution over n events. Sampling stops as soon as the distance drops below a user-defined threshold. Distance functions such as Kullback-Leibler, Chi-square, and Hellinger distance have been used [RFS03b].

Recently, Kuznetsov et al. [KKR18] proposed a sampling map estimator network to increase sample density at locations where a subsequent denoising network yields unsatisfactory results otherwise. Hasselgren et al. [Has+20] extend this approach by considering the temporal domain. The denoised image of the previous frame is reprojected with optical flow warping and then passed to the sampling map estimator as well as the denoising network. In doing so, samples of previous frames can be reused such that the sampling process can focus on previously occluded regions. For further adaptive super-sampling techniques in Monte Carlo rendering, we refer the reader to the survey of Zwicker et al. [Zwi+15].

2.7. Explainable Artificial Intelligence

In today's world, artificial intelligence (AI) has a significant impact on our everyday life. It curates information presented to us, guides us in making decisions, or even makes the decisions for us. For many mundane tasks, society has accepted that most AI systems deployed today lack transparency in their decision making, that is we do not understand how an AI system comes to a certain conclusion. However, this is unacceptable in safety-critical environments such as autonomous driving or medical diagnosis. With the recent and omnipresent success of DL, transparency becomes, even more, an issue since DL models operate on a multitude of abstract, high-dimensional feature spaces, with their presentation in a human-understandable form being difficult as best. Explainable artificial intelligence (XAI) recognizes and addresses the aforementioned limitations of AI by providing tools for reasoning about AI decisions. For an extensive overview of XAI's philosophy and numerous works in this field, we refer to the survey of Adadi and Berrada [AB18].

Generally speaking, there are two options to achieve transparency. Either one designs transparent AI systems that operate on human-understandable signals, or ways have to be found to make signals in existing AI approaches interpretable. Inference engines are a good example of the first option. They serve as the backbone of various expert systems. However, they tend to be quite limited in their expressive power. In contrast, black-box DL systems are highly effective, but not well understood. Thus, recent research—including our work about inverse feature visualization [RW20]—has focused on implementing the second option to induce transparency in DL.

Pixel Attribution Pixel attribution methods, also called saliency maps, intend to explain the outcome of an image processing deep neural network (DNN) by attributing each input pixel according to its influence on a specific output. By inspecting highly influential regions in the (human-understandable) input image, one can obtain indicators about the impact of certain entities on a network’s decision. Thus, pixel attribution methods provide explanations on a global scale, by highlighting *what* is important to the network. However, these methods usually do not provide information *why* something is important. Although pixel attribution mostly has been applied to image classification networks, many methods presented here can be readily adapted to other input and output domains. As long as both domains can be communicated in a human-understandable format, attributions can be readily used to generate interpretable decision indicators.

Several works generate pixel attributions by introducing modifications to the input. By obfuscating parts of the input image and observing the effects on the network output, influential regions can be identified. Zeiler and Fergus [ZF14] generate a heatmap of attributions by sliding a grey occluder over the input image. Fong and Vedaldi [FV17] blur local regions of the image or apply random noise to them. Ribeiro et al. [RSG16] decompose the input image into superpixels and find a limited set of them that activates a specific class neuron the most.

Pixel attribution methods based on deconvolution invert the flow of information in a CNN. After providing an input image, the activation signal at the neuron of interest is masked and propagated back to the input by providing “inverse” formulations for each preceding layer. Zeiler and Fergus [ZF14] propose to invert max-pooling operations by masking all locations that do not contribute to a maximum value. Convolutions are inverted by applying the transpose of their linearization, see Sec. 5.2. Several modifications of this approach exist, such as Grad-CAM by Selvaraju et al. [Sel+17], or the approach of Springenberg et al. [Spr+14] that leaves out pooling layers. However, there is no clear theoretical evidence that inversion operations yield meaningful pixel attributions. In particular, Mahendran and Vedaldi [MV16] show that pixel attributions from deconvolution approaches lack distinguishability when computing selections for different neurons at deeper levels, making the overall deconvolution approach questionable.

Other pixel attribution methods exploit a theoretically grounded way of backpropagating information by following the gradient of the output signal. Given an activation signal of interest a , the assumption is that a highly influential pixel p results in a high gradient $\partial a / \partial p$. This approach is motivated by the observation that high gradients indicate a high sensitivity of a w.r.t. p in a local environment around p . Thus, Simonyan et al. [SVZ13] propose to obtain pixel attributions by backpropagating the gradient signal through the network to obtain gradients regarding the input image. However, this approach comes with two issues. First, gradients of neural networks tend to be highly sensitive to small input perturbations [GAZ19], and they are spatially noisy. Smilkov et al. [Smi+17] thus propose to average gradients over multiple inputs that are perturbed by some Gaussian noise. Second, a high gradient may be a sufficient condition for highly influential regions, but it is by no means a necessary one. For instance, if the value of p is chosen such that it maximizes the activation a , changing p significantly will have a large impact on a . The gradient $\partial a / \partial p$ however is zero due to the necessary condition for optimality. This issue is known by the name model saturation. To address it, Shrikumar et al. [SGK17] propose to backpropagate finite differences of activations regarding a reference image instead. Similarly, Sundararajan et al. [STY17] introduce a reference image and then use it to integrate the gradient over the interpolation path between the reference and the investigated image. Sturmfels et al. [SLL20] investigate how the choice of a reference image influences the pixel attributions returned by Sundararajan’s method.

All pixel attribution methods have in common that they are difficult to verify. Verification mostly has been conducted by investigating pixel attributions for a set of samples and checking if the result seems plausible. However, this approach runs into the danger of only accepting methods that show what we consider to be meaningful, while methods showing what *is* meaningful may be discarded. Although it is still an open question how to identify “good” pixel attribution methods, misleading approaches have been identified by running specific sanity checks. Adebayo et al. [Ade+18] invalidate some widely deployed pixel attribution methods by checking if attributions are similar for correctly trained networks, untrained ones, or networks that have been trained on a wrong label set. Ghorbani et al. [GAZ19] test the robustness of pixel attribution methods by optimizing for very small ℓ^∞ -bounded noise patterns on the input such that the network output remains constant, but pixel attributions change significantly. Kindermans et al. [Kin+19] show that some methods are not invariant to constant shifts in the input data if they are compensated for by updating the bias term in the first network layer.

Feature Visualization Pixel attribution methods can only highlight signals that are present in a user-specified input image. As a consequence, any such approach is always biased towards a specific dataset. In the worst case, analysis is performed over a dataset that does not reflect the input domain the network was trained for. Then, many salient structures of neurons will not be available. The re-

sulting pixel attributions will either be arbitrary, or worse, misleading. Further, pixel attributions may become problematic when investigating hidden neurons deep inside a DNN. In contrast to neurons in the output layer, they are likely to be sensitive to rather general concepts that appear at multiple locations in the image. Thus, attributions become fragmented and difficult to interpret.

Feature visualization approaches the problem of XAI differently. Instead of searching for salient structures in a prescribed input, salient structures of a neuron are synthesized on the fly by performing a process called activation maximization [Erh+09]. Given a user-defined neuron in a DNN, activation maximization optimizes for an input image that maximally activates the chosen neuron. The optimized image is called a (feature) visualization of the specific neuron. Various feature visualization techniques and their applications have been surveyed by Nguyen et al. [NYC19].

By design, feature visualizations do not depend on a specific dataset and thus do not suffer from the aforementioned limitations. However, they come with other limitations regarding interpretability. In contrast to pixel attributions that highlight structures in presumably human-understandable images, there are no guarantees that feature visualizations returned by activation maximization depict interpretable structures. As it turns out, feature visualizations mostly show high-frequency noise patterns if no regularization is enforced during the optimization process. Although these kinds of patterns may represent a valid feature learned by a network—we discuss this later when considering adversarial examples—they are not very helpful for understanding neural networks.

To improve upon the interpretability of feature visualizations, several regularizations have been proposed. Simonyan et al. [SVZ13] apply a L_2 regularization term during optimization. Yosinski et al. [Yos+15] suggest blurring feature visualizations during optimization to reduce high-frequency noise patterns. They also clip low-impact pixels to obtain a cleaner image. Mahendran et al. [MV15] introduce a total variation regularization to directly penalize high-frequency patterns during optimization. Nguyen et al. [Ngu+16] utilize an image prior learned by a pre-trained image generator network. Feature visualizations then are optimized in the latent feature space of the generator instead of the RGB pixel space. The prior ensures that images roughly resemble human-understandable concepts. However, features that were not seen during the training of the generator network cannot be reproduced. Thus, this approach is biased towards the training dataset of the generator. Olah et al. [OMS17] propose a strong regularizer by randomly jittering, rotating, and scaling the input during optimization. Thus, only feature visualizations are generated for which the visualized neuron is robust against the aforementioned operations. Additionally, they propose to optimize images in Fourier space to obtain better descent directions during optimization.

Feature visualizations are not limited to single neurons. One can also perform activation maximization jointly for groups of neurons. Olah et al. [OMS17] use this insight to visualize channels in a CNN. Similarly, all neurons of a layer can be visualized jointly [MOT15] for artistic purposes, or to obtain an impression of the granularity of learned features. Recently, feature visualizations have

been used extensively by a group of collaborators to reverse-engineer GoogLeNet [Cam+20]. They found out that GoogLeNet learns rotationally equivariant features, a rich set of curve features, high-low frequency detectors, and many more. They provide a taxonomy of feature classes per layer and present indications that similar features form across multiple models and tasks.

Adversarial Examples In 2013, Szegedy et al. [Sze+13] observed that DNNs can be easily manipulated to form wrong decisions by adding a small, imperceptible noise pattern to the image. They synthesize these noise patterns with a technique similar to activation maximization, by performing a first-order optimization of the input w.r.t. a wrongly labeled loss function. The resulting, perturbed images are called adversarial examples, as they exploit the deficiencies of a specific network. In principle, one needs access to the network gradients to “attack” a network and generate adversaries. Hence, this approach is called a white-box attack. However, adversarial examples also generalize to other models than the one being attacked. For this reason, black-box attacks can be performed by running a white-box attack on a network under own control, and then deploying the same adversaries against the “foreign” network. The existence of adversarial examples may seem counterintuitive at first. However, this is not very surprising when considering how many search directions for adversaries exist in the high-dimensional input domain. Adversarial examples demonstrate that we are lacking intuition for high-dimensional spaces (see Sec. 2.3), whereas a network is able to detect correlations even in spurious high-dimensional data [Goh19]. For the same reason, we conjecture that noisy, unregularized feature visualizations may be representative of a specific neuron.

As it turns out, generating adversarial examples is inexpensive. Goodfellow et al. [GSS14] observe that a single optimization step that updates each pixel by a small constant times the sign of its gradient suffices to generate potent adversaries. This approach is called the fast gradient sign method. Kurakin et al. [KGB18] show that adversaries can be strengthened further by repeated updating according to the fast gradient sign method. They also propose targeted attacks by using the fast gradient sign method to decrease a loss function favoring a user-specified, but wrong outcome. Further, adversarial examples occur in various settings. They are not limited to spatially incoherent perturbations. Alcorn et al. [Alc+19] show that adversarial examples can be found by rendering objects under random translations and rotations. Further, adversarial examples with realistic, physically plausible perturbations can be created from a differentiable renderer by manipulating environmental lighting and mesh vertex positions [Liu+18].

Robust Training Adversarial examples indicate that the goal of XAI to provide interpretable insights into DNNs might be ill-posed. Any plausible, human-understandable explanation of a DNN would have to explain both adversaries and “friendly” input alike. However, the existence of indistinguishable adversaries implies that any XAI method must be able to explain multiple contradicting

network outputs for conceptually identical input. In doing so, the method appears to be opportunistic and—similarly to some pixel attribution methods—thus questionable.

Robust training intends to counteract adversarial examples by designing specific optimization routines that mitigate adversary-enabling effects. Szegedy et al. [Sze+13] improve on network robustness against adversarial examples by mixing them into the dataset during training. Goodfellow et al. [GSS14] suggest incorporating the “one step”-adversary of the fast gradient sign method into the loss function to improve network resilience. This modification requires two backpropagation passes instead of one per training step, and thus increases network resilience at the cost of a 50% performance decrease when training. The idea of considering adversarial examples in the loss function led to formalizing robust training as a saddle-point problem [Mad+17]. That is, one intends to find the optimal network parametrization such that the maximal attainable loss due to a perturbation-bounded adversary is minimized. In practice, Madry et al. [Mad+17] run two nested optimization routines to solve the saddle-point problem. The inner routine performs multiple steps of projected gradient descent to find the “worst-case” adversary in an epsilon ball around the actual input data. The outer routine minimizes the loss via a standard gradient descent step performed on the adversary returned by the inner routine. As a result, saddle-point optimization can be considered as standard training with adversarial data augmentation. We note, however, that training time increases by several multiples compared to standard loss minimization.

Instead of integrating adversarial examples directly into the loss formulation, one can also regularize the network’s gradient w.r.t. the input, so that the sensitivity of the network regarding small input perturbations is reduced. For instance, Ross and Doshi-Velez [RD18] introduce a regularization term that promotes small gradients. However, to evaluate the regularization term, an optimizer requires access to the Hessian of the loss. Cisse et al. [Cis+17] introduce Parseval networks that are Lipschitz continuous with a Lipschitz constant of one. Consequently, Parseval networks guarantee that small epsilon ball modifications do not have a large impact on the network output.

Similarly, Wong et al. [WK17; Won+18] guarantee that, given a user-prescribed epsilon ball for which perturbations are permitted, the network output does not leave a specific region, the convex outer bound of the adversarial polytope. In particular, they propose an algorithm to compute bounds of linear optimization problems that are constrained to this region. Last but not least, Papernot et al. [Pap+16] show that robust classifiers can be obtained by training against class probability vectors as returned by another network trained on the standard dataset.

Robust training commonly lowers accuracy on the original dataset since additional constraints have to be considered during optimization. On the upside, by applying robust training, one can train DL models that do not suffer from adversarial examples and thus have a higher chance of being amenable to interpretation. Thus, robustly trained networks can be seen as a step towards designing interpretable AI systems. Indeed, it has been observed by Tsipras et al. [Tsi+18] that pixel attributions

returned by gradient-based methods align notably better with human-understandable concepts when being applied to robust networks. Engstrom et al. [Eng+19] show that if applied to robustly trained networks, activation maximization yields high-quality feature visualizations without the need for any priors such as transformation robustness or image regularization. In particular, they show that feature visualizations remain consistent when varying the initial image that subsequently is optimized by activation maximization. Similarly, Roberts and Tsiligkaridis [RT21] observe that feature visualizations with hard L_1 and L_2 constraints show easy-to-interpret shapes as long as visualizations are generated for robust networks. Additionally, they utilize perturbations introduced by adversarial examples over robustly trained networks to indicate which features in an image are important for classification.

Parameterized Splitting of Summed Volume Tables

Summed-area tables (SATs) by Crow [Cro84] are considered a convenient acceleration structure to quickly compute integral values over arbitrary rectangular regions in a 2D data array F . They are realized by replacing the value at index (x, y) in the array with precomputed sums over all values contained in the rectangular subarray that is spanned by the indices $(1, 1)$ and (x, y) , that is

$$\text{SAT}_F[x, y] := \sum_{x' \leq x, y' \leq y} F[x', y'].$$

Partial sums of F for arbitrary rectangular subarrays can be computed in constant time, by making use of the inclusion-exclusion principle. Instead of reading and adding up values of F along the whole region spanned by $(x_1 + 1, y_1 + 1)$ and (x_2, y_2) , it suffices to read the SAT-values at the corners of the selected subarray. It holds that

$$\sum_{x_1 < x' \leq x_2, y_1 < y' \leq y_2} F[x', y'] = \text{SAT}_F[x_1, y_1] + \text{SAT}_F[x_2, y_2] - \text{SAT}_F[x_1, y_2] - \text{SAT}_F[x_2, y_1]$$

with $\text{SAT}_F(x, y)$ set to zero if $x = 0$ or $y = 0$. Thus, SATs reduce the summation of $(x_2 - x_1) \cdot (y_2 - y_1)$ values to a summation of four values. The concept of SATs extends to any number of dimensions.

Given a d -dimensional array F , the corresponding d -dimensional summed-volume table (SVT) is realized by

$$\text{SVT}_F[v_1, v_2, \dots, v_d] := \sum_{v'_i \leq v_i} F[v'_1, v'_2, \dots, v'_d].$$

By the inclusion-exclusion principle, it holds that

$$\sum_{v_i < x'_i \leq w_i} F[x'_1, x'_2, \dots, x'_d] = \sum_{\lambda \in \{0,1\}^d} p(\lambda) \text{SVT}_F[\lambda_1 v_1 + (1 - \lambda_1) w_1, \dots, \lambda_d v_d + (1 - \lambda_d) w_d] \quad (3.1)$$

where $p(\lambda) = (-1)^{\sum_{i \in \mathbb{N}_{\leq d}} \lambda_i}$ is the parity of λ and $(v_i)_{i \in \mathbb{N}_{\leq d}}, (w_i)_{i \in \mathbb{N}_{\leq d}}$ are multi-indices with $v_i \leq w_i$. As a consequence, partial sums of hyperboxes (line segments in 1D, rectangles in 2D, cuboids in 3D, and so on) can be computed by evaluating a SVT at the 2^d corner points of that hyperbox. In the special case of $d = 1$, a SVT stores prefix sums of a 1D array. This motivates our convention of calling entries in a SVT d -dimensional prefix sums, or simply prefix sums by omitting the dimension.

Limitations of Summed-Volume Tables Since prefix sums are obtained by summing over large regions of data, the precision required to accurately store them exceeds the precision of F . In the most extreme case, F is a bit field with n entries, requiring a total of n bits of storage, whereas the SVT of F must be able to store entries of non-binary integers with maximal value n . Storing the SVT in fixed precision requires at least $n \cdot \lceil \log_2(n + 1) \rceil$ bits. For large n , SVTs cannot be stored in environments with tight memory constraints. For instance if $n = 2048^3$, F takes up 1GB of data and fits into VRAM of most graphics processing units (GPUs). On the other hand, SVT_F requires 34GB, a demand that—as of yet—cannot be satisfied by most GPU models. In the future, it is expected that this memory gap increases further since dataset sizes n tend to scale linearly with available RAM capacity. That is, whereas available memory grows linearly in n , SVTs grow in $n \log n$.

As is noted by Hensley et al. [Hen+05], the aforementioned precision limitations become even more obvious when deploying SVTs with floating-point arithmetic. If the dynamic range of the data array is high, computing partial sums via the inclusion-exclusion principle suffers from catastrophic cancellation. In the worst case, a single outlier stored at the beginning of the dataset corrupts the whole SVT. If it is several magnitudes larger than the rest of the data, all prefix sums take on the same outlier value due to rounding errors in the floating-point IEEE 754 representation. Any partial sum computed via Eq. (3.1) evaluates to zero, independently of the actual data values stored in the respective hyperbox. For this reason, we do not suggest to implement SVTs over floating-point values at all. In the case of continuous data values with a low dynamic range, data first can be rescaled and quantized to integers with minimal loss. The integral values then can be used to build the SVT. Negative entries in the integral representation can be eliminated by adding $-\min(F)$ to each value. Without loss of generality, it can be assumed that F stores non-negative integral numbers.

In the following, we give an overview of our SVT data structure [RW21a]. It addresses the large memory footprint of classical SVTs by performing recursive split operations as explained in Sec. 3.1. The properties of our data structure are presented in Sec. 3.2. Further, in Sec. 3.3, we describe a heuristic that derives SVT representations for fixed memory or performance budgets. Insights that

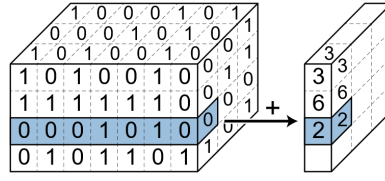


Figure 3.1.: Computation of the first three entries of the aggregate array by summing up values along the split dimension. For the third entry, summed values are highlighted in blue. Figure has been taken from our presentation held at EuroVis 2021 [RW21b].

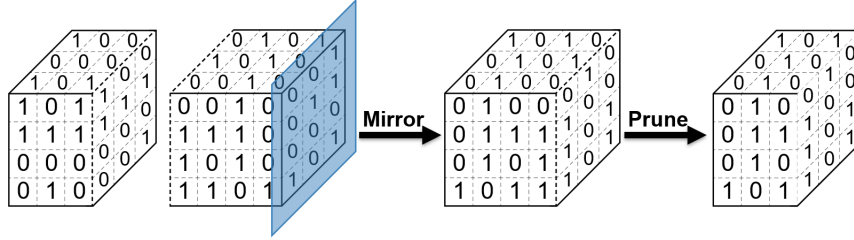


Figure 3.2.: Application of the conjugate trick to the block of Fig. 3.1. After separating the front and rear half, the rear half is mirrored and pruned. Figure has been taken from our presentation held at EuroVis 2021 [RW21b].

were recognized after composing the papers are discussed in Sec. 3.4. Last, alternative representations of others are discuss in Sec. 3.5. Here, we also show that most alternatives are special cases of our flexible approach.

3.1. SVT Split Operation

We require the following notation. An array F is said to have shape $n \in \mathbb{N}^d$ if and only if it is d -dimensional of shape $n_1 \times n_2 \times \dots \times n_d$. We define the size of n by $|n| := \prod_{i=1}^d n_i$. In particular, an array of shape n has $|n|$ elements. Further, for any multi-index $v \in \mathbb{N}^d$ and integers $k \in \{1, \dots, d\}$, $i \in \mathbb{N}$, we denote by $v|_{k=i}$ the multi-index that is obtained by replacing the k -th component of v by i . When accessing array elements via a multi-index, $F[v]$ is a shorthand notation for $F[v_1, \dots, v_d]$.

The central idea behind our data structure is the process of splitting the array F of shape $n \in \mathbb{N}^d$ into a small array F_a of precomputed, high precision aggregates and a set $\{F_{s_0}, F_{s_1}, \dots\}$ of low precision subarrays such that any prefix sum can be efficiently computed from one prefix sum of the aggregate array and one prefix sum of a single subarray. By recursively splitting the data F into smaller blocks of partial sums, at some point, blocks become sufficiently small such that the extra logarithmic factor required to store the SVT of a block does not cause significant memory overhead.

The split process is initiated by choosing a split dimension $k \in \mathbb{N}_{\leq d}$. Along this dimension, ℓ axis-aligned cut planes $H_i = \{x \in \mathbb{R}^d \mid x_k = c_i\}$ are placed at integral split positions $c_1 < c_2 < \dots < c_\ell$

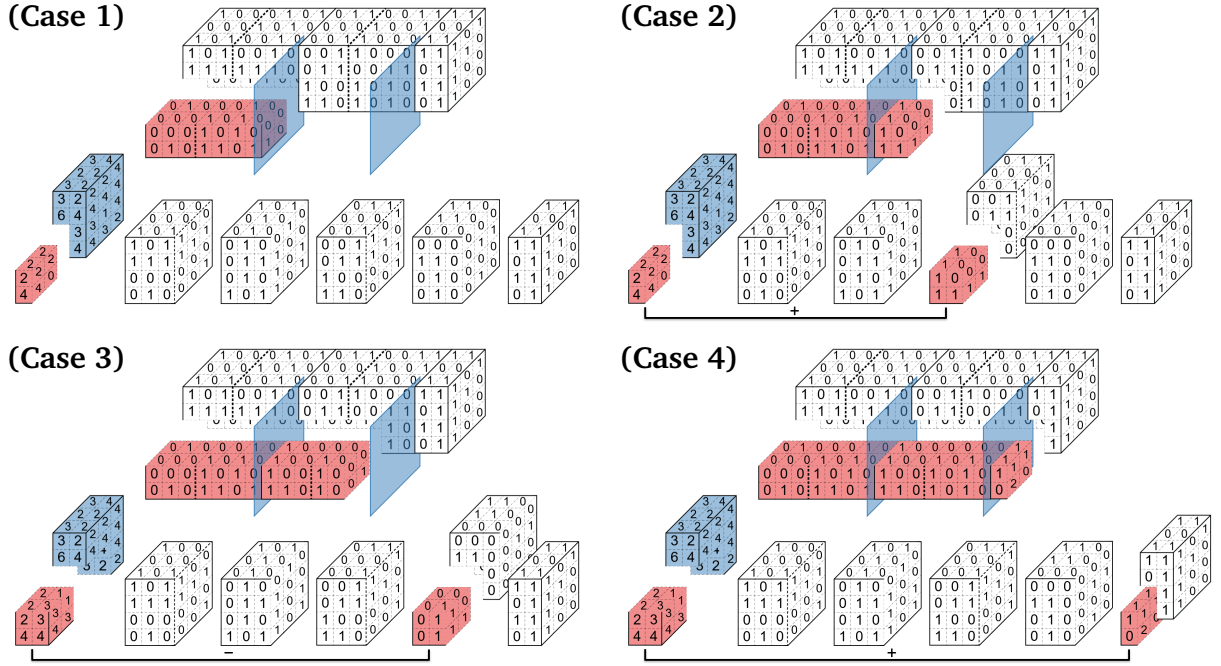


Figure 3.3.: All cases arising when computing prefix sums of the original array F (top block) by querying prefix sums of arrays obtained after splitting (lower blocks). Cut planes, as well as the aggregate array, are colored in blue. The additional split of each block into a front and a rear part is indicated by a dashed line. Red blocks indicate the regions over which prefix sums are considered. For each case, the prefix sum of the top block can be recovered by combining up to two prefix sums located in the bottom blocks. Figure has been taken from our presentation held at EuroVis 2021 [RW21b].

(see blue planes in Fig. 3.3). After splitting F along the cut planes, each block except for the last one is processed as follows: First, we populate a $(d - 1)$ -dimensional hyperslice by summing up all values of the block along the split dimension, see Fig. 3.1. Second, the block is processed according to the conjugate trick, see Fig. 3.2. For this, we place an additional cut in the middle of the block and add the front half to the list of low precision subarrays. The rear half is flipped along the split dimension, and then its last hyperslice is removed. The resulting array is again added to the list of subarrays. After processing each block, 2ℓ subarrays and ℓ hyperslices of aggregated values are obtained. The high precision aggregate array F_a is obtained by stacking the hyperslices along the split dimension. The last block is added without modification to the list of subarrays. A splitting process thus creates one aggregate array F_a and $2\ell + 1$ subarrays.

Now, a prefix sum of F can be efficiently computed from at most one prefix sum of the aggregate array and one prefix sum of a single subarray. Let ν be a multi-index. We distinguish four cases, see Fig. 3.3. In the first case, one has $\nu_k = c_i$ for a suitable $i \in \mathbb{N}_{\leq \ell}$. Then, $\text{SVT}_F[\nu]$ equals the prefix sum $\text{SVT}_{F_a}[\nu |_{k=i}]$ of F_a . In the second case there exists $i \in \mathbb{N}_{\leq \ell}$ with

$c_i < v_k \leq \lfloor (c_{i+1} + c_i)/2 \rfloor$, so that v is located in the front half of a block. Then it holds that $\text{SVT}_F[v] = \text{SVT}_{F_a}[v \mid_{k=i}] + \text{SVT}_{F_{s_{2i}}}[v \mid_{k=v_k-c_i}]$. In the third case v is located in the rear half and we find $i \in \mathbb{N}_{\leq \ell}$ such that $\lfloor (c_{i+1} + c_i)/2 \rfloor < v_k < c_{i+1}$. The prefix sum is computed by subtracting a prefix sum from the mirrored rear half as follows: $\text{SVT}_F[v] = \text{SVT}_{F_a}[v \mid_{k=i+1}] - \text{SVT}_{F_{s_{2i+1}}}[v \mid_{k=c_{i+1}-v_k}]$. Lastly, if v falls into the last block, i.e. $v_k > c_\ell$, then the prefix sum is computed similarly to the second case via $\text{SVT}_F[v] = \text{SVT}_{F_a}[v \mid_{k=\ell}] + \text{SVT}_{F_{s_{2\ell}}}[v \mid_{k=v_k-c_\ell}]$.

Values in SVTs of subarrays arise as sums over only a fraction of values of F . They require fewer bits of precision than values in the SVT of F . Hence, at the cost of one additional data-fetch operation per prefix sum, the memory footprint of a SVT is reduced by storing SVTs of the aggregate arrays and subarrays respectively. In particular, the memory savings can be reinforced at the cost of more fetches by recursively applying the split process to the newly acquired arrays F_a and all F_{s_i} until a certain termination condition holds. Then, each terminal array is stored by encoding either its entries verbatim or the entries of its SVT in fixed precision [RW21a].

Parameter Tree The positions c_i of cut planes can be chosen in such a way that each subarray has one out of two distinct shapes. By assuming that all subarrays with equal shapes are processed similarly, we can express the memory layout of our data structure through a ternary tree, called parameter tree, where internal nodes encode split operations and external nodes encode write operations to memory. More importantly, the tree allows us to quickly compute the memory footprint of our data structure as well as an upper bound for the number of data-fetch operations required to compute a prefix sum. The details can be found in our publication.

3.2. Properties of the Data Structure

The proposed data structure exhibits a couple of properties that are important for applications. First and foremost, by controlling how the parameter tree is constructed, the memory footprint can be freely adapted between $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$. At the same time, the number of data-fetch operations required to retrieve a prefix sum varies between $\mathcal{O}(1)$ and $\mathcal{O}(n)$. In general, memory consumption can be decreased by increasing the number of fetches and vice versa. In contrast to the work of Urschler et al. [UBD13], the memory footprint of our data structure does not depend on the actual values stored in F .

In our paper, we show that our data structure can be constructed in $\mathcal{O}(n)$. Decoding and updating values is achieved by traversing the parameter tree in a top-down fashion and tracking the blocks in which multi-indices are located (see four cases in Sec. 3.1). Note that in each split operation, the block index can be computed from the current multi-index by a simple div-operation. Thus, the cost of

computing the locations of memory access is negligible compared to the cost of fetching the memory itself. For an exhaustive comparison of our properties to other works, we refer to Table 2 in our paper.

3.3. Heuristic for Building Parameter Trees

Given a fixed budget of either memory consumption or the number of data-fetch operations, the optimal parameter tree minimizing the respective other, unconstrained quantity can be found by solving a combinatorial optimization problem. By implementing a Branch-and-Bound algorithm, we were able to identify optimal parameter trees for binary arrays of shape $64 \times 64 \times 64$ in 12.5 hours. Clearly, this approach does not scale for arrays of a larger shape. Instead, we propose a heuristic that is derived from inspecting the optimal trees of shape $64 \times 64 \times 64$. It is implemented in such a way that it returns a parameter tree $\mathcal{H}(\lambda, n)$ requiring at most λ data-fetch operations for an array F of shape n . If $\lambda = 1$ or $\lambda \geq |n|$, the optimal parameter tree trivially is given by a single node indicating whether F or SVT_F is stored in fixed precision. In any other case, an internal node splitting along the largest dimension is generated. Cut planes are placed according to a handcrafted function taking in the required number of fetches λ . Similarly—by utilizing a second handcrafted function—a fixed amount of λ_a data-fetch operations is allocated for querying the aggregate array. The remaining $\lambda - \lambda_a$ operations are allocated to the subarrays. At this point, the shapes and data-fetch constraints are known for the aggregate array as well as the subarrays after splitting. Hence, one can invoke the heuristic function \mathcal{H} recursively to determine the subtrees that are going to be attached to the internal node. The derivation of handcrafted functions can be found in Sec. 1 of the supplementary material provided with our paper.

Although the heuristic is designed for constraining the number of data-fetch operations, it can be utilized for memory constraints by performing a bisection search over λ . First, the lower bound of data-fetch operations is set to one, and an upper bound of data-fetch operations is found by invoking the heuristic for increasing powers of two until a parameter tree satisfying the memory constraint is returned. Then, interval nesting is performed. In each step, the heuristic is invoked at the midpoint $(\lambda_{\min} + \lambda_{\max})/2$. If the returned parameter tree violates the memory constraint, subsequent searches are restricted to the upper half of the current search interval. Otherwise, the lower half is chosen.

3.4. Addendum to the Paper

After publishing our work, we realized that timings for evaluating the heuristic can be reduced significantly by replacing distributed aligned splits with `at_end` aligned splits (see Sec. 4.1 of the paper). The shapes of arrays represented by the third child of each internal node become smaller so that overall less recursive invocations of \mathcal{H} are required to arrive at the termination conditions. Further,

we would like to mention here that the inaccuracy of the heuristic at $\lambda = 4$ can be compensated for by brute-forcing the optimal parameter tree for $\lambda \leq 4$ whenever the bisection search stops at $\lambda = 5$. A brute-force approach is reasonable in this scenario since any parameter tree with at most 4 data-fetch operations is limited to at most three internal nodes. Also, we note that our representation can be used in tandem with anchoring the prefix sum origin at the center of the data array, as suggested by Hensley et al. [Hen+05] of. If doing so, one can save an additional number of d bits per SVT entry by storing separate SVTs for each hyperoctant of the data array.

3.5. Alternative Representations of Summed-Volume Tables

In the following, we establish the basics of reference methods we compare against. We point out that most approaches are special cases of our approach. They can be recovered by considering specific parameter trees that may or may not make use of the conjugate trick. In particular, any optimal parameter tree outperforms those reference methods by definition. As shown in the paper, the same holds for parameter trees returned by the proposed heuristic.

Partial Summed-Volume Tables First, we discuss partial SVTs as introduced by Zellmann et al. [ZSL18]. They propose to chunk a 3D data array into bricks of 32^3 and store a SVT over 32^3 elements per brick. Thus, each SVT entry can be represented by a 16bit unsigned integer. When computing partial sums along regions that do not fit into a single 32^3 brick, one value has to be fetched from memory for each brick intersecting the queried region. Since there still are $\mathcal{O}(n/(32^3)) = \mathcal{O}(n)$ many bricks, this approach scales equally poorly as summing up all values by iterating over the input array directly. To circumvent this issue, Zellmann et al. propose to build a hierarchical representation of partial SVTs by taking the last SVT entry of each brick and arranging them in an array of shape $n_1/32 \times n_2/32 \times n_3/32$. This array again is bricked and summed up partially. However, the authors admit that all bricks that overlap only partially with the queried region have to be processed at their current hierarchy level. Hence, the number of touched bricks reduces to the size of the region boundary. In the best case of cuboid-like data arrays, the complexity still is $\mathcal{O}(n^{2/3})$. If the data array has one flat dimension, that is, there exists $i \in \mathbb{N}_{\leq d}$ such that $n_i \ll n_j$ for all $j \neq i$, complexity remains in $\mathcal{O}(n)$. Thus, we compare against the non-hierarchical version only.

Note that the bricking approach of Zellmann directly generalizes to arbitrary dimensionality and that they can be represented by our recursive split operations if the conjugate trick during splitting is omitted. That is, instead of creating two subarrays per block, we only create a single subarray that is derived from the processed block by removing its last hyperslice. The respective parameter tree is visualized in Fig. 3.4.

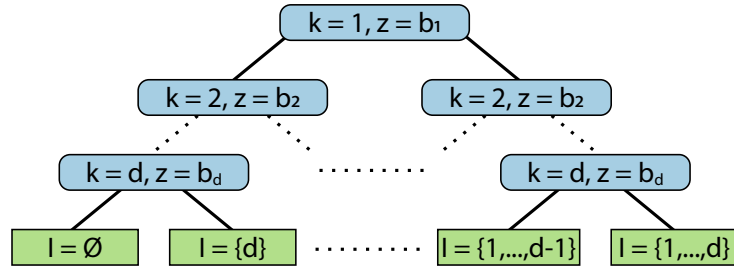


Figure 3.4.: Parameter tree describing partial SVTs by arranging the data array into bricks of shape $b \in \mathbb{N}_{\leq d}$. Please refer to our paper for explanations of node parameters. We assume that the shape n of the original data array is a multiple of b , that is $n_i = m_i b_i$ for an appropriate choice of $m \in \mathbb{N}_{\leq d}$. Then, all subarrays of a split operation have equal shapes, so that the parameter tree reduces to a binary tree with 2^d leaves. By splitting into blocks of length b_i at dimension b_i , we ensure that no sums spanning multiple bricks are computed. If a path to a specific leaf navigates into a subarray node—i.e. turns right—when splitting along dimension i , we add i to the set I of dimensions indicating along which dimensions array values are cumulated. If the path navigates into the aggregate array node, no further summation along dimension i is required. It is already performed when constructing the aggregate array. By design, arrays at the leaf notes can be stored with a fixed precision of $\lceil \log_2 |b| \rceil$.

Generalization of the Approach of Ehsan et al. Ehsan et al. [Ehs+15, Sec. 7.2] propose a technique to compute arbitrary prefix sums by reading one value from the original data array and three values from a reduced set of prefix sums containing only 5 out of 9 SAT entries. Thus, memory requirements of SAT decrease by 44%. Their approach generalizes to arbitrary numbers d of dimensions by only storing every third hyperslice of the SVT so that 2^d out of 3^d SVT entries are not required anymore. As a consequence, the modified data array now can be viewed as a collection of 3^d shaped bricks for which prefix sums are not known at the 2^d corner voxels. If an unknown prefix sum has to be computed, one can always find multi-indices v, w with $w_i = v_i + 1$ such that the left side of Eq. (3.1) equals to the data value $F[w]$ and the right side sums over the unknown prefix and $2^d - 1$ known prefix sums. After rearranging Eq. (3.1) such that the unknown prefix sum is isolated, it can be computed from $2^d - 1$ known prefix sums and one data value.

The conjugate trick as proposed in our paper can be considered an even more general reformulation of the approach of Ehsan et al., in which a) dimensions are decoupled from each other by defining the split operation over a single split dimension, and b) one is not limited to considering every third hyperslice. Instead, arbitrary spacings between considered hyperslices can be assumed. In particular, the d -dimensional generalization of Ehsan’s approach can be described by the parameter tree shown in Fig. 3.5

Fenwick Trees Fenwick trees were first introduced in 1994 [Fen94] as a compact data structure with $\mathcal{O}(n)$ memory that enables the computation of arbitrary prefix sums over 1D arrays of length n in $\mathcal{O}(\log n)$ time. After applying a slight variation suggested by Schneider et al. [SR17], they can

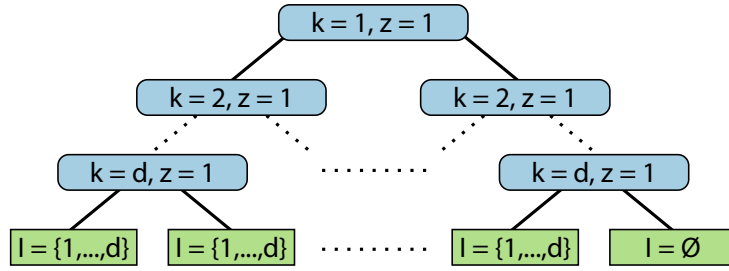


Figure 3.5.: Parameter tree representing the technique of Ehsan et al. [Ehs+15]. Please refer to our paper for explanations of node parameters. Note that all subarrays have a side length of 1 in direction of the split dimension so that the parameter tree reduces to a binary tree with 2^d leaves. If a path to a specific leaf navigates into an aggregate array node at least once—i.e. turns left—then the values stored at the leaf are located on the hyperslices carrying SVT entries. Thus, I is set to $\{1, \dots, d\}$. The unique leaf that is reached when following subarray links only encodes the data values appearing on the left side of Eq. (3.1). Hence, we set $I = \emptyset$.

be realized by repeated application of a simple lifting concept in which the first value of each pair of array entries is stored in memory, and the second value is summed up with the first one to form a new array of length $n/2$. This process is repeated for the newly acquired array, up to the point where no entry is left over.

As shown by Mishra in 2013 [Mis13], Fenwick trees can be deployed for multidimensional arrays as well. By considering d -dimensional arrays as 1D arrays of $d - 1$ -dimensional hyperslices, the process described above can be executed with hyperslices as entries. Whenever a hyperslice is about to be stored in memory, we compute its $(d - 1)$ -Fenwick tree representation and store this instead. At some point in this recursive process of “peeling dimensions”, hyperslices reduce to zero-dimensional arrays (i.e. scalars), which then are stored in memory as is. Mishra also proves that arbitrary prefix sums are computed in $\mathcal{O}(\prod_{i \in \mathbb{N}_{\leq d}} \log n_i)$, and Schneider et al. [SR17] show later in 2017 that memory consumption is linear in $|n|$.

The recursive nature of both lifting and dimension peeling aligns perfectly with our recursive splitting approach. A parameter tree with $\prod_{i \in \mathbb{N}_{\leq d}} n_i$ leaves describing multidimensional Fenwick trees can be obtained by unrolling the directed acyclic graph as shown in Fig. 3.6 and omitting the conjugate trick.

Memory Efficient Integral Volume Memory efficient integral volume (MEIV) by Urschler et al. [UBD13] reduces the memory footprint of 3D SVTs by fitting one-parameter models to bricks of data, and then storing the model parameters in fixed precision and the remaining error in a dynamic word length buffer. First, the full SVT is computed and chunked into bricks of small size. For each brick with low corner v , the lowest entry $\text{SVT}[v]$ as well as an integral model parameter μ is saved.

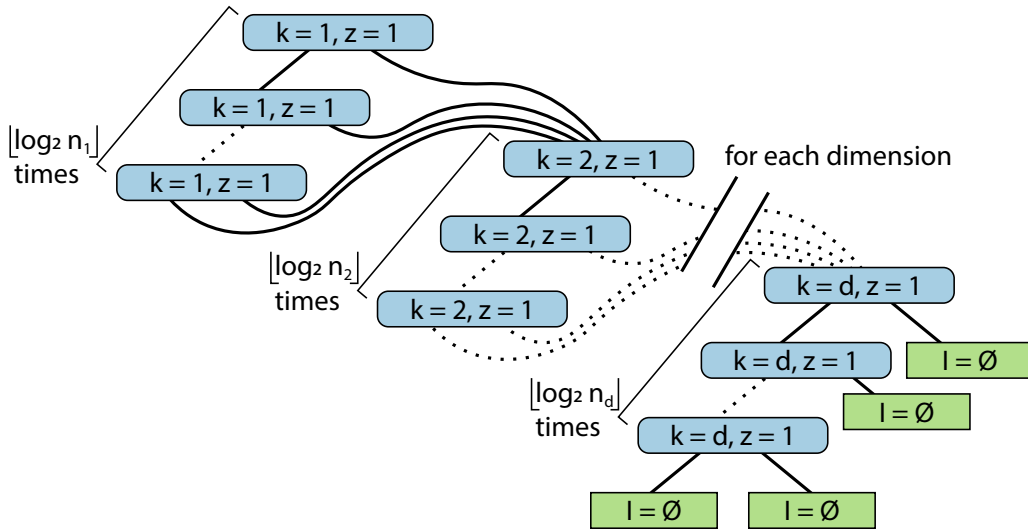


Figure 3.6.: Directed Acyclic Graph of the parameter tree describing d -dimensional Fenwick trees. Similar subtrees were summarized to ease visual inspection. Please refer to our paper for explanations of node parameters. Note that the parameter tree reduces to a binary tree with $\prod_{i \in \mathbb{N}_{\leq d}} \lfloor \log_2 n_i \rfloor$ leaves since all subarrays have a sidelength of 1 in direction of the split dimension.

Here, μ is chosen such that the bit lengths of all remaining errors

$$e[w] = \text{SVT}[v] + \mu \cdot \left(\prod_{i \in \mathbb{N}_{\leq d}} (w_i + 1) - \prod_{i \in \mathbb{N}_{\leq d}} (v_i + 1) \right) - \text{SVT}[w],$$

where w is a multi-index located in the current brick, are minimized. The authors suggest determining μ via binary search. Lastly, the remaining errors $e[w]$ are stored in a dynamic word length storage with the smallest precision possible such that all errors of the current block can be encoded. In doing so, the authors exploit a) that errors $e[w]$ tend to be notably smaller than the prefix sums $\text{SVT}[w]$ and b) that errors and prefix sums alike are of smaller magnitude for bricks of small multi-indices.

MEIV decreases memory consumption exceptionally well. At the same time, a prefix sum $\text{SVT}[w]$ can be computed from two data fetches from memory. It suffices to query brick-related information ($\text{SVT}[v]$, μ), as well as the error $e[w]$ from the dynamic word length storage. The clear downside of MEIV is the increased construction time due to fitting μ . In the authors' experiments, building the MEIV representation took up to 75 times longer than building the regular SVT. In contrast to our approach, MEIV cannot give any memory guarantees before encoding the actual data. Ill-posed input may yield sufficiently higher memory consumption than observed in the authors' work. This observation also implies that no parameter tree can be found that reflects the behavior of MEIV.

Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques

Dimensionality reduction techniques are utilized for mapping high-dimensional point sets to low-dimensional embeddings such that structural properties (i.e. clusters, topology) of points are retained. In doing so, structures can be explored and analyzed using 2D visualization tools. However, popular dimensionality reduction techniques such as multidimensional scaling and t-distributed stochastic neighbour embedding produce different 2D embeddings for each run. Whereas many structures consistently appear over multiple runs of a specific dimensionality reduction technique, some structures may be spurious when only inspecting a single run. Therefore, to obtain reliable insights into the data, we propose a visualization technique that allows—at a glance—to identify regions in which most 2D point sets coincide and thus can be considered reliable [RKW20].

Before presenting our contribution in Sec. 4.6 and Sec. 4.7, we provide information about existing algorithms used throughout our work. First, the dimensionality reduction techniques that have been used to generate the input to our method are explained. Second, we discuss Voronoi diagrams and HDBSCAN clustering, which both play a central role in the data processing pipeline for generating robustness plots. We also introduce the k-Means and DBSCAN clustering algorithms. They are used to derive the member-centric cluster variability plots [Kum+18] against which we compare our method. Last, we discuss the algorithms utilized to select representative embeddings, that is matrix seriation and the PageRank algorithm.

4.1. Dimensionality Reduction

Dimensionality reduction (DR) techniques intend to map data points embedded in a space of arbitrary dimension n into a low-dimensional space of dimension $m < n$. If m is chosen to be two, points reduced via DR can be explored in 2D visualizations, facilitating new insights into the original point

set. Formally, let $\mathcal{X} = \{x_i\}_{i \in \mathbb{N}_{\leq |\mathcal{X}|}} \subset \mathbb{R}^n$ be finite set of points. Then, a DR technique maps each point x_i to a low-dimensional representative $y_i := f(x_i) \in \mathbb{R}^m$. Ideally, the mapping function f is able to preserve as much of the original points' structure as possible. For instance, pairwise distances in \mathcal{X} may be preserved, that is $\|x_i - x_j\| \approx \|f(x_i) - f(x_j)\|$. As a consequence, clusters and other topological properties can be estimated by inspecting the low-dimensional point embedding $f(\mathcal{X})$. Inevitably, if the intrinsic dimension of \mathcal{X} is larger than m , some structural properties of \mathcal{X} will not be reflected in $f(\mathcal{X})$. For instance, a set of $n + 1$ pairwise equidistant points can be embedded faithfully in n dimensions, but not less, as then the equidistance property would be lost. The phenomenon that low dimensional spaces cannot provide sufficient space to reflect certain point configurations is known as the crowding problem. Depending on the DR technique, the crowding problem can be mitigated by focussing on either global or local inter-point relationships, at the cost of inaccurate representation of the respective other quantity.

t-Distributed Stochastic Neighbour Embedding t-distributed stochastic neighbour embedding (t-SNE) [MH08] favors embeddings that accurately represent point neighborhoods and thus can be considered a local DR technique. At its core, t-SNE models the neighborhoods of high-dimensional points and low-dimensional representatives separately as two discrete probability distributions P and Q , which then are matched in an iterative optimization scheme minimizing the Kullback-Leibler divergence between P and Q .

In high-dimensional space, the probability distribution of x_i is obtained by placing a normalized Gaussian kernel of fixed variance σ_i centered at x_i so that the probability of walking from x_i to y_j is given by

$$p_{j|i} := \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma_i^2))}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / (2\sigma_i^2))}.$$

For each point, σ_i is determined via binary search such that $\lambda = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$ holds, where λ is a user-chosen perplexity parameter. It is considered a smooth measure for counting the effective size of the neighborhood of x_i modeled by t-SNE and is commonly set to values between 5 and 50. The joint probability distribution P encoding all high-dimensional neighborhoods is obtained by symmetrizing and renormalizing the conditional probabilities, that is $p_{ij} := (p_{j|i} + p_{i|j}) / (2|\mathcal{X}|)$.

On the low-dimensional end, neighborhoods are modelled via a Student t-distribution

$$q_{ij} := (1 + \|y_i - y_j\|^2 / \nu)^{-(\nu+1)/2} / Z \quad \text{with} \quad Z := \sum_{k \neq \ell} (1 + \|y_k - y_\ell\|^2 / \nu)^{-(\nu+1)/2},$$

where the t-SNE authors propose to set $\nu = 1$ for $m \leq 3$ and $\nu > 1$ otherwise. After initializing the representatives y_i with random positions in \mathbb{R}^m , the Kullback-Leibler divergence $\text{KL}(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$ is iteratively decreased by a customized gradient descent optimizer

that incorporates an adaptive learning rate scheme and a progressively increasing momentum term. By design, the outcome of t-SNE heavily depends on the choices for λ , ν and—in particular—the initial configuration of the low-dimensional representatives. To mitigate the randomness of t-SNE, its authors propose to include an early exaggeration step in the initial stages of the optimization to separate point clusters and converge to a reasonable global configuration. Nonetheless, t-SNE’s non-deterministic nature persists.

Barnes-Hut t-SNE Evaluating the gradient of the Kullback-Leibler divergence

$$\frac{\partial \text{KL}}{\partial y_i} = 4 \left(\sum_{j \neq i} Z p_{ij} q_{iq} (y_i - y_j) - \sum_{j \neq i} Z q_{ij}^2 (y_i - y_j) \right) \quad (4.1)$$

requires $\mathcal{O}(|\mathcal{X}|^2)$ time, making the application of t-SNE unfeasible for large point sets. With a couple of approximations, the complexity of t-SNE is decreased to $\mathcal{O}(|\mathcal{X}| \log |\mathcal{X}|)$ [Maa14]. First, the high-dimensional probability distribution P is thinned out by setting all $p_{j|i}$ to zero for which x_j is not one of the $\lfloor 3\lambda \rfloor$ nearest neighbors of x_i . The remaining $\lfloor 3\lambda \rfloor$ -neighborhoods are efficiently identified by running depth-first searches in a Vantage-point tree over \mathcal{X} . Thus, the first sum in Eq. (4.1) is reduced to a constant number of terms.

Second, the low-dimensional representatives are structured in an m D-hyperoctree, that is a quadtree for $m = 2$ and an octree for $m = 3$. Given a representative y_i and a threshold θ trading off speed for accuracy, the largest non-empty cells \mathcal{C} of side length r_{cell} and center y_{cell} in the hyperoctree are identified for which $r_{\text{cell}} < \|y_i - y_{\text{cell}}\|^2 \theta$ holds. By exploiting that all points of a cell are almost equally distant to y_i , the second term in Eq. (4.1) can be approximated by

$$\sum_{\text{cell} \in \mathcal{C}} Z N_{\text{cell}} q_{i,\text{cell}}^2 (y_i - y_{\text{cell}}), \quad (4.2)$$

where $q_{i,\text{cell}} = (1 + \|y_i - y_{\text{cell}}\|^2 / \nu)^{-(\nu+1)/2} / Z$ and N_{cell} is the number of representatives falling into the respective cell. With the same trick of grouping up points in sufficiently large cells, Z can be computed in $\mathcal{O}(|\mathcal{X}| \log |\mathcal{X}|)$. In particular, with Z available, $q_{i,\text{cell}}$ can be computed in $\mathcal{O}(1)$, so that Eq. (4.2) can be computed in $\mathcal{O}(\log |\mathcal{X}|)$ for each y_i .

Scaling by Majorizing a Complicated Function Multidimensional scaling (MDS) [KW78; CC08] is a set of techniques for finding a mapping f such that $\|x_i - x_j\| \approx \|f(x_i) - f(x_j)\|$ is as tight as possible for all points. In comparison to t-SNE, pairs of points do not have to be close neighbors in order to be influential on f . Thus, MDS can be considered a global DR technique. The scaling by majorizing a complicated function (SMACOF) algorithm proposed by De Leeuw [LM09] is a specific

MDS approach we use in our implementations. It minimizes

$$g(Y) := \sum_{i < j} (d_{ij}(X) - d_{ij}(Y))^2$$

where $X \in \mathbb{R}^{|\mathcal{X}| \times n}$ and $Y \in \mathbb{R}^{|\mathcal{X}| \times m}$ are the matrices with rows x_i respectively y_i , and $d_{ij}(Z)$ is given by the euclidean distance between the i -th and j -th row of a matrix Z .

Instead of directly solving for $\arg \min_Y g(Y)$, SMACOF assumes a random start configuration $Y^{(0)}$ and then repetitively solves for $Y^{(i+1)} = \arg \min_Y \bar{g}(Y, Y^{(i)})$ where \bar{g} is a majorant of g than can be optimized for quickly. Specifically, it is

$$\bar{g}(A, B) := \text{trace}(X^T V X) + \text{trace}(A^T V A) - 2 \text{trace}(A^T Z(B) B)$$

with

$$V = \sum_{i < j} (e_i - e_j)(e_i - e_j)^T \quad \text{and} \quad Z(B)_{ij} = \begin{cases} d_{ij}(X)/d_{ij}(B), & d_{ij}(B) > 0 \\ 0, & d_{ij}(B) = 0 \end{cases}.$$

By a simple reformulation of g and the Cauchy-Schwartz inequality stating $\|x\| \cdot \|y\| \geq |x^T y|$ with equality taken on if $x = y$, it can be shown that $g(Y) = \bar{g}(Y, Y)$ and $g(Y) \leq \bar{g}(Y, Z)$ for all $Z \in \mathbb{R}^{|\mathcal{X}| \times m}$. We refer the reader to the MDS review by Groenen and Van De Velden for the details [GV16, pp. 7–8]. Now, it follows that

$$g(Y^{(i+1)}) \leq \bar{g}(Y^{(i+1)}, Y^{(i)}) \leq \bar{g}(Y^{(i)}, Y^{(i)}) = g(Y^{(i)}),$$

in other words, decent over g is guaranteed. At some point, SMACOF converges to $Y^{(i+1)} \approx Y^{(i)}$. Then, $Y^{(i)}$ contains the positions of the low-dimensional representatives. Similar to t-SNE, the result of SMACOF heavily depends on the initial configuration $Y^{(0)}$.

4.2. Voronoi Diagrams

Assume a finite set of points $P \subset \mathbb{R}^2$. The Voronoi cell associated with $p \in P$ is given by

$$\{x \in \mathbb{R}^2 \mid \|x - p\| \leq \|x - q\| \text{ for all } q \in P\},$$

that is all points that share p as nearest neighbor in P . The set of all Voronoi cells forms the Voronoi diagram. It partitions the ambient space \mathbb{R}^2 , and it permits a binary relationship over P by calling two points of P adjacent if and only if their corresponding Voronoi cells share a common facet. k -order Voronoi diagrams contain Voronoi cells defined over k -subsets of P . They group all points in \mathbb{R}^2 that share the closest k nearest neighbors in P . That is, for a k -subset V of P , its associated Voronoi cell is

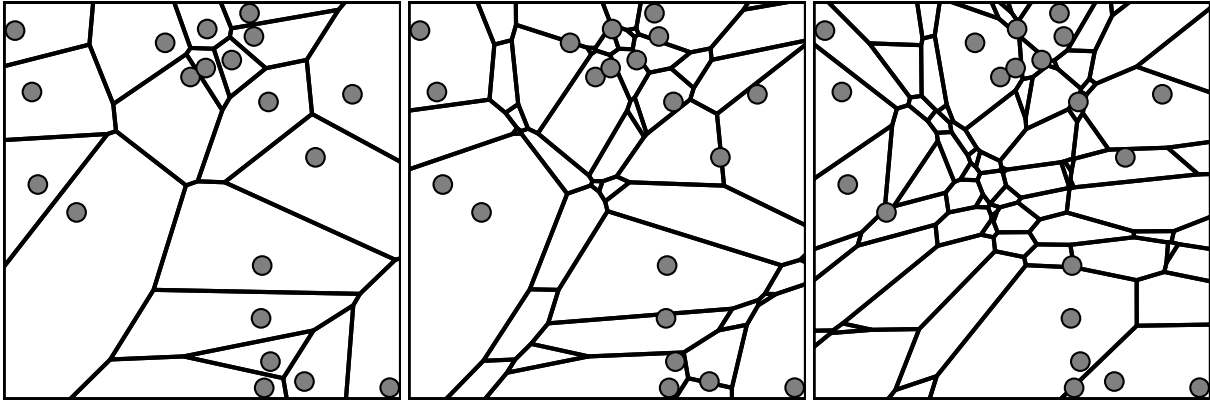


Figure 4.1.: Voronoi diagrams of order 1, 2 and 5 (left to right) for a point set of 20 randomly sampled points over \mathbb{R}^2 . Point positions are indicated by circles. Cells are separated by ridges.

given by

$$\{x \in \mathbb{R}^2 \mid \|x - p\| \leq \|x - q\| \text{ for all } p \in V, q \in P \setminus V\}.$$

Sample Voronoi diagrams of various orders are depicted in Fig. 4.1. Note that not all k -subsets of P give rise to a non-empty Voronoi cell. In general, k -order Voronoi diagrams contain only $\mathcal{O}(k|P|)$ non-empty Voronoi cells, whereas there exist $\mathcal{O}(|P|^k)$ k -subsets in P . Further, a k -order Voronoi diagram defines a binary relationship over k -subsets of P by requiring a common facet for Voronoi cells of adjacent subsets. Thus, the k -subsets of P can be equipped with a graph structure.

Computation of Voronoi Diagrams The dual graph of a 1-order Voronoi diagram over P is the Delaunay triangulation of P , which can be computed from a convex hull in \mathbb{R}^3 . For this reason, one can use the quickhull algorithm [BDH96] to first compute a 3D convex hull and then use this result to derive the 1-order Voronoi diagram. Arbitrary k -order Voronoi diagrams can be computed from 1-order Voronoi diagrams by following an algorithm of Lee [Lee82]. It gradually generates Voronoi diagrams of increasing order by cutting cells of the previous order $k-1$. As the $k-1$ nearest neighbors for a cell are already known, the remaining k -th neighbor is computed by intersecting the cell with the 1-order Voronoi diagram obtained when discarding the $k-1$ known neighbors. Then, the resulting cutouts with similar k -neighborhoods are stitched together.

The run-time complexity of computing k -order Voronoi diagrams with the aforementioned algorithm is in $\mathcal{O}(k^2|P| \log |P|)$, which is reasonable for modest values of k . However, Voronoi computation can be sped up significantly by utilizing the GPU [SKW09]. By discretizing the 2D plane to a high-resolution grid and propagating closest neighbors of grid cells in four sweeps, Voronoi diagrams can be computed quickly on the GPU's SIMD architecture, with runtime being linear in the resolution of the grid.

4.3. Clustering

Clustering commonly is used in data mining to condense objects with similar characteristics into a single “cluster group” that is representative of all objects it contains. This approach is in line with the intuition of humans processing data. We tend to quickly categorize large collections of objects before reasoning about them, as otherwise our mental model would be overburdened. The resulting categories—that is clusters—then are considered entities and can be handled more easily due to their modest quantity. Similarly, clustering in data processing eases the design of subsequent algorithms. For instance, Kumpf et al. [Kum+18] cluster points according to their position in space, so that different point embeddings can be compared by matching a small set of clusters between embeddings. We utilize clustering to assemble large stable point subsets from smaller k -subsets, which in turn are utilized to derive a cleaned-up visualization of k -order Voronoi diagrams.

k-Means k-Means can be considered as one of the most widespread clustering algorithms available and dates back to a work of Steinhaus in 1956 [Ste+56]. It is very effective in detecting circular-shaped clusters in large point sets P by minimizing its distance to k centroids that can be freely positioned in space. That is, k-Means intends to solve

$$\arg \min_{c_1, c_2, \dots, c_k} \sum_{p \in P} \min_i \|p - c_i\| \quad (4.3)$$

After choosing a random initial position for all centroids, the following two-step process is applied to update centroid positions. First, the clusters

$$C_i = \{p \in P \mid \|p - c_i\| < \|p - c_j\| \text{ for all } j \neq i\}$$

are determined by assigning each point $p \in P$ to its closest centroid. Then all centroids are updated by setting $c_i := (\sum_{p \in C_i} p) / |C_i|$. This process is repeated until cluster assignments and thus centroids do not change anymore. Albeit being simple in nature, k-Means suffers from multiple drawbacks. As k-Means ends up in local optima of Eq. (4.3), the quality of found solutions highly depends on the initial configuration. k-Means usually takes a lot of iterations to reach convergence, and it even may not converge at all for certain initial configurations. Also, it is sensitive to outliers in P . Lastly, k-Means does not provide guidance on how to choose the number of clusters k . Many improvements and adaptations on k-Means have been suggested over the years, many of which can be found in the survey of Filippone et al. [Fil+08].

Density-Based Spatial Clustering of Applications with Noise In comparison to k-Means, density-based spatial clustering of applications with noise (DBSCAN) does not search for clusters

centered at specific points but instead follows connected regions of densely packed objects [Est+96]. It thus is suited for identifying submanifold structures of any shape. Its behavior is controlled by two user-defined parameters $\epsilon \in \mathbb{R}$ describing the maximal distance for which points are considered neighbors, and $\eta \in \mathbb{N}$ defining the number of points when a region is considered dense. Given a point set P and a randomly chosen member $p \in P$, DBSCAN considers its ϵ -neighborhood

$$\mathcal{N}_\epsilon(p) = \{q \in P \mid \|p - q\| < \epsilon\}.$$

A new cluster C containing p and $\mathcal{N}_\epsilon(p)$ is initiated if $|\mathcal{N}_\epsilon(p)| \geq \eta$. As long as there exists a point $q \in C$ with $|\mathcal{N}_\epsilon(q)| \geq \eta$, the cluster is grown by adding all points in $\mathcal{N}_\epsilon(q)$ not yet assigned to any other cluster. When there are no new points to add, the cluster is complete. Further clusters are generated by the same scanning process initialized at randomly chosen, not yet clustered points $p \in P$ with $|\mathcal{N}_\epsilon(p)| \geq \eta$. DBSCAN ends when all points $p \in P$ with $|\mathcal{N}_\epsilon(p)| \geq \eta$ have been assigned to a cluster. Any point that is not clustered at that point is considered noise. It neither has a dense neighborhood nor is located close to an existing cluster.

On the plus side, DBSCAN can detect arbitrary cluster shapes and is robust against outliers. Efficient implementations are realized by deploying acceleration structures for finding close neighbors of points. Further, the number of clusters is not prescribed and will be derived on the fly by DBSCAN. On the downside, the results of DBSCAN heavily depend on the order of picking initial cluster positions, as well as the choice of ϵ and η . Whereas η usually can be derived from the context in which clustering is performed, choosing a suitable ϵ is challenging at best. In the case of datasets with spatially varying density levels, a global value for ϵ cannot be chosen such that it suits all parts of the dataset equally well.

Hierarchical DBSCAN Campello et al. [CMS13] propose a variant of DBSCAN in which the ϵ parameter is dropped in favor of a hierarchical cluster representation, thus the name hierarchical density-based spatial clustering of applications with noise (HDBSCAN). Intuitively, the proposed algorithm identifies clusters for all values of ϵ and arranges them in a merge tree according to their subset relationships. This tree further is tagged with a temporal component at each junction that tracks at which ϵ -value two clusters merge into a larger one. In particular, each cluster is assigned a “lifetime” by computing the difference between the ϵ -value where it comes into existence due to a merge event, and the ϵ -value where it is incorporated into an even larger cluster by merging with another cluster. The resulting merge tree then is traversed in bottom-up order to identify a disjunct set of clusters for which the product of cluster size and lifetime becomes maximal. HDBSCAN thus favors large clusters of locally high densities. That is, the points contained in a cluster are significantly

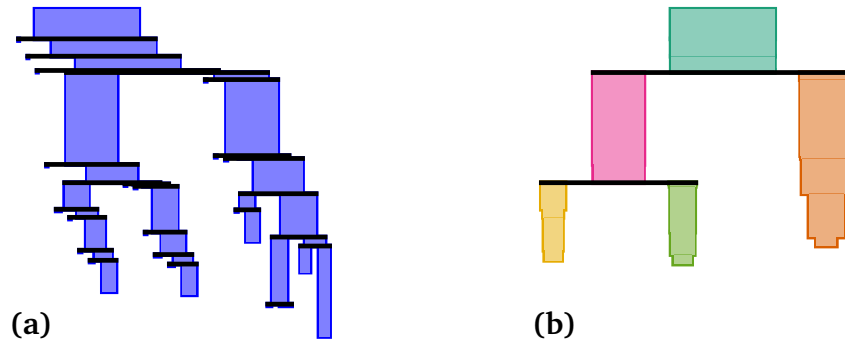


Figure 4.2.: Dendrograms visualizing an exemplary cluster hierarchy generated by HDBSCAN (a) before and (b) after the simplification process. Each box indicates a cluster with the width encoding its size. Horizontal lines signal merge events. Figure has been adapted from Fig. 4 in [RKW20].

more dense than spurious points in the surrounding. Similarly to DBSCAN, noise points are identified according to the η parameter already introduced in the previous section.

HDBSCAN implementations proceed in several steps. First, the core distance of each point $p \in P$ is defined by setting $d_{\text{core}}(p) := \min_{\epsilon} |\mathcal{N}_{\epsilon}(p)| \geq \eta$. It is computed via a nearest neighbor search. The core distance defines the minimal ϵ -value such that p can enter a cluster-relationship with other points. Thus, two points p, q can only be arranged in a cluster if the ϵ -value exceeds their mutual reachability distance

$$d_{\text{mreach}}(p, q) := \max\{d_{\text{core}}(p), d_{\text{core}}(q), \|p - q\|\}.$$

In the second step, single-linkage clustering with respect to d_{mreach} is performed. In other words, a hierarchical cluster representation is generated by running Prim's algorithm [Pri57] on the fully connected, undirected graph over P with edges weighted according to d_{mreach} . Initially, all points are considered to be singleton clusters. Whenever Prim's algorithm connects two vertices p and q via the edge $d_{\text{mreach}}(p, q)$, the clusters previously containing p and q respectively are merged. In the merge tree, this is reflected by introducing a new node with its children set to the nodes representing the merged clusters. Additionally, the temporal ϵ -value of the node is set to $d_{\text{mreach}}(p, q)$.

At this point, the merge tree still contains a lot of small, spurious clusters, compare Fig. 4.2a. To remove noise, all nodes of clusters being smaller than η_{cluster} are dropped. In particular, a lot of nodes of the merge tree are left with a single child. Such a node is integrated into its parent node by relinking its child to the grandparent and adding annotations describing the integrating node's cluster size and ϵ -value. After this simplification, a node does not represent a single cluster anymore, but rather a progression of a cluster collecting surrounding points as ϵ increases. The resulting simplification of the cluster hierarchy can be observed in Fig. 4.2b. Commonly, η_{cluster} is set to η to avoid the introduction of a new user-chosen parameter.

As the last step, a flat clustering is obtained from the simplified cluster hierarchy. To do so, each node in the merge tree is assigned a persistence value that is computed by integrating the cluster size over ϵ . The final clustering is given by the optimal disjunct set of clusters such that the sum of their persistence values becomes maximal. It is computed in a single bottom-up sweep through the simplified merge tree. The maximal attainable persistence value for each subtree is computed as the maximum of the root node's persistence and the sum over child persistence. The corresponding, optimal set of clusters is set to either the singleton set containing the root node cluster, or the union of optimal cluster sets as returned by the child nodes. The final output of HDBSCAN then can be queried from the root node of the merge tree.

4.4. Matrix Seriation

Heatmaps are a common visualization tool for displaying (dis-)similarities between two sets of entities, or pairwise (dis-)similarities between a single set of entities. Depending on the order in which entities are assigned to rows and columns, correlations and clusters between entities become visible. However, without carefully ordering entities, correlations are difficult to spot, see Fig. 4.3. Matrix seriation addresses this issue by finding a permutation of columns and rows such that highly correlated entities are displayed next to each other and structures such as clusters become apparent. Specifically, we deploy the rank-two ellipse seriation algorithm of Chen [Che02]. The survey of Behrisch et al. [Beh+16] suggests that it is well suited to reveal blocks along the diagonal, where each block hints at a separate cluster. Further, its implementation is rather straightforward. Starting with the (unsorted) initial distance matrix $R^{(0)}$, matrices $R^{(i+1)} := \phi(R^{(i)})$ are iteratively computed by invoking the Pearson correlation operator

$$\phi(A)_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} \cdot C_{jj}}}$$

where C is the covariance matrix of A given by

$$C = \frac{1}{|\mathcal{X}|} MM^T \quad \text{with} \quad M_{ij} = A_{ij} - \frac{1}{|\mathcal{X}|} \sum_k A_{ik}.$$

Chen noticed that with an increasing number of iterations, the matrix rank decreases and the columns of $R^{(i)}$ tend towards the surface of an ellipsoid. Hence, when stopping the iterative process at a matrix rank of 2, the columns of $R^{(i)}$ can be ordered as they appear on the 2D ellipsis. This is easily achieved by computing 2D coordinate vectors of the columns w.r.t. the plane spanned by the two-dimensional image of $R^{(i)}$, and sorting coordinate vectors according to their polar angles. The resulting permutation can be applied to the rows and columns of the original distance matrix $R^{(0)}$ to disclose clusters, see Fig. 4.3b.

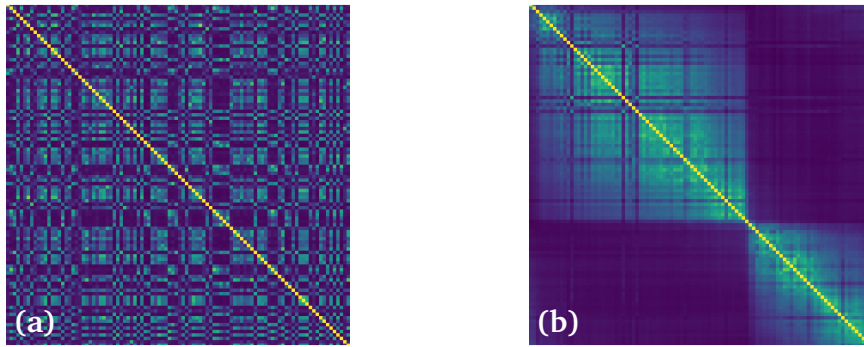


Figure 4.3.: Sample pairwise similarity matrix (a) before and (b) after ordering point sets according to the rank-two ellipse seriation algorithm. Note that after sorting, it becomes evident that point sets can be assigned to a total of two groups. This information cannot be immediately deduced from (a), although the same data is shown. Figure has been adapted from Fig. 6 in [RKW20].

4.5. PageRank

Originally, the PageRank algorithm has been deployed to assign scores to web pages returned by a web crawler. It is designed in such a way that web pages linked from many other pages receive a higher score and that links emanating from highly scored pages have a higher impact on the target page as well. Mathematically, PageRank is formulated as a Markov process and solved via the eigenvalue problem $M\lambda = \lambda$ where λ represents the scores assigned to each page and M_{ij} represents the probability of a user on page j following a link to page i . In practice, the PageRank algorithm incorporates an additional dampening factor d (defaulting to $d = 0.85$) and solves for the eigenvalue problem $\lambda = (dM + \frac{1-d}{|\mathcal{X}|} \mathbf{1}\mathbf{1}^T)\lambda$. This scheme avoids ill-posed solutions and speeds up the convergence of the Von Mises iteration used to solve the eigenvalue problem.

4.6. Robustness Plots

Now, we present our contribution. As discussed in Sec. 4.1, t-SNE and SMACOF require an initial configuration of representatives y_i that subsequently is refined in an iterative optimization process. By executing a DR technique multiple times for a fixed set \mathcal{X} of high-dimensional points and random low-dimensional initializations, an ensemble \mathcal{E} over embeddings of \mathcal{X} into the 2D plane is created. Here, each ensemble member $P \in \mathcal{E}$ is an ordered 2D point set in which each point refers to a unique entity in \mathcal{X} . In particular, points between different ensemble members are identified with each other if they refer to the same high-dimensional point.

The ensemble \mathcal{E} serves as input to our method. After picking a distinguished point set $P \in \mathcal{E}$, our method analyzes its k -neighborhoods and identifies subgroups of points for which neighborhood relationships are consistent across most 2D point sets in \mathcal{E} . Then, those subgroups are visualized

in a novel robustness plot. In theory, one can now generate a separate robustness plot for each ensemble member and visualize all of them in a large dashboard. However, such a visualization quickly overburdens the user. Since it is expected that many 2D point sets of the ensemble are related to each other, we propose to cluster the ensemble and only show one representative robustness plot for each cluster.

k -Neighborhood Analysis From now on, we assume that a point set $P \in \mathcal{E}$ and a number $k \in \mathbb{N}$ have been fixed. Given a k -subset $V \subset P$, we compute per ensemble member $Q \in \mathcal{E}$ the smallest possible disk that covers the points being identified with V . The stability value of V with respect to Q is then computed as the ratio between k and the number of points covered by the disk. By averaging the stability values over all ensemble members, we obtain a scalar value that represents the stability of V with respect to the whole ensemble. Thus, we consider a k -subset to be stable if, for most ensemble members, its corresponding points are close to each other without any other interfering points in between. Since computing stability values for all $\binom{|P|}{k} \in \mathcal{O}(|P|^k)$ k -subsets is intractable for point sets of modest size, we suggest inspecting the subsets that arise as non-empty cells in the k -order Voronoi diagram of P . As pointed out in Sec. 4.2, this reduces the number of subsets to $\mathcal{O}(k|P|)$, which is linear in the number of points. In addition to that, the k -order Voronoi diagram provides a meaningful spatial relationship between k -subsets that we subsequently use to cluster stable k -subsets into larger subsets of locally stable points. As is elaborated in our publication, this would not have been possible when only inspecting $|P|$ subsets that arise from each point of P and its $k - 1$ nearest neighbors.

Clustering Stable Subgroups of Points To detect stable subsets with more than a limited number of k points, we propose to cluster the Voronoi cells of investigated k -subsets with a slightly modified version of the HDBSCAN algorithm (see Sec. 4.3). First, we consider Voronoi cells as objects that are to be clustered. The core distance of each cell is given by $1 - \lambda^2$, where λ is the stability value as described above. The distance measure $\|\mathcal{C} - \mathcal{D}\|$ between two cells \mathcal{C}, \mathcal{D} is given by zero if \mathcal{C} and \mathcal{D} share a common facet, and by infinity otherwise. That is, two Voronoi cells can only merge if they are adjacent to each other. Second, we alter the notion of cluster size. Since we ultimately aim for clustering the point set P , and since k -order Voronoi cells are not in one-to-one correspondence with the points in P , we introduce a significance relationship between points and Voronoi cells. Here, each point has a significance of one, which then is distributed between all Voronoi cells the point is related to. The significance of a Voronoi cell is given by the sum of all received significance contributions from its related points. Instead of measuring the size of a Voronoi cell cluster by counting the number of its cells, we now sum up the significance values of all contained cells. Please refer to our paper for a detailed discussion about our choice of the core distance and significance contributions.

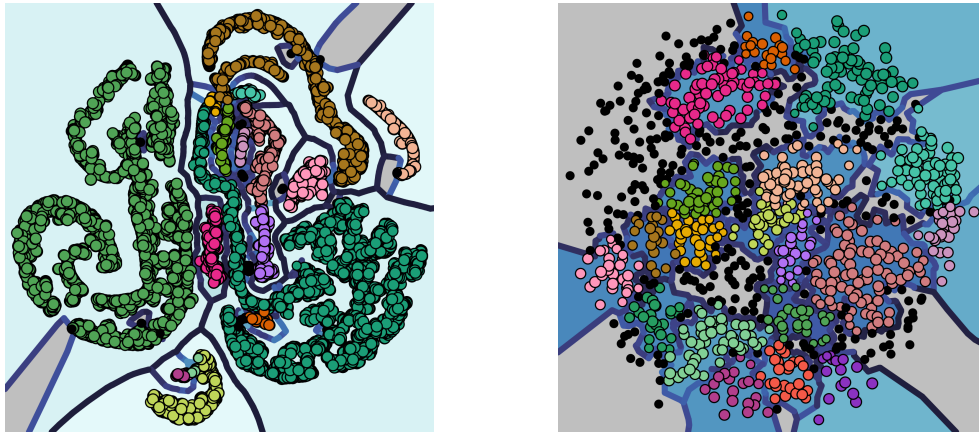


Figure 4.4.: Exemplary robustness plots for two different point set ensembles \mathcal{E} . Figure has been adapted from Fig. 1 in [RKW20].

After computing a Voronoi cell clustering with HDBSCAN, a postprocessing step is applied to extract clusters of stable points. By making use of the significance relationship between points and Voronoi cells again, Voronoi cell clusters can be transformed into fuzzy clusters over P . A final defuzzifying operation yields point clusters that represent subgroups of points with stable k -neighborhood relationships.

Visualizing Stable Subgroups of Points We visualize the clusters of stable subgroups through a space-filling tessellation of the 2D plane. The tessellation is determined by first computing a 1-order Voronoi diagram of P and then merging all adjacent cells where points belong to the same cluster. In particular, we map the stability value of a subgroup to the color of its corresponding region. Further, regions are separated by ridges, with ridge color indicating the separation strength of adjacent regions. To provide context, we also render a scatterplot of P on top of the tessellation where points are colored according to cluster membership. Two exemplary robustness plot are shown in Fig. 4.4. Note that black points and gray regions stem from noise detected by HDBSCAN. They indicate high instability, or in other words, neighborhood relationships in gray regions do not persist across the point set ensemble \mathcal{E} .

4.7. Representative Point Sets

Since showing a robustness plot per point set in \mathcal{E} would overburden the visualization, we guide the user's selection process of viewing certain point sets by identifying groups of similar point sets as well as a suitable representative per group. The pairwise similarity between point sets P and Q encodes how well the k -neighborhoods arising in Q are preserved in the point set P . It is computed

by aggregating the stability value of V with respect to P over all subsets V that occur in the k -order Voronoi diagram of Q . Then, we plot all pairwise similarities in a heatmap where a single row or column corresponds to a point set in the ensemble. Further, rows and columns are ordered according to the rank-two ellipse seriation algorithm (see Sec. 4.4), so that clusters of points sets are easily discernible, see Fig. 4.3b.

After the user selects a subset $\mathcal{E}_{\text{cl}} \subset \mathcal{E}$ from the heatmap, the PageRank algorithm (see Sec. 4.5) is utilized to find the most representative point set in \mathcal{E}_{cl} . Let S be the matrix containing the pairwise similarities of point sets in \mathcal{E}_{cl} . Then, we define the matrix M as follows.

$$M_{ij} = \frac{(1 - \delta_{ij})S_{ij}}{\sum_{k \neq i} S_{kj}}.$$

It is a stochastic matrix where M_{ij} encodes the probability of the j -th point set choosing the i -th as its “most similar” neighbor in the ensemble. By solving the (dampened) eigenvalue problem of M , one obtains an impact score for each point set in \mathcal{E}_{cl} , with the highest scoring point set being elected as the representative for which we show a robustness plot. Note that we consider PageRank to be a suitable choice in this context since PageRank penalizes dissimilar, outlier point sets to have overall less influence on scoring results.

The upcoming chapters 6 and 7 make heavy use of basic deep learning concepts. Albeit not being substantial in the context of the previous work presented in Chap. 4, even there some deep learning was involved when generating high dimensional datasets. In the following, we present basic concepts about training procedures (Sec. 5.1) and neural network architectures (Sec. 5.2) required to fully comprehend our work.

Neural networks are highly universal function approximators in the sense that—given enough adjustable network parameters—arbitrary functions g can be implemented by fitting a neural network on sufficiently many function samples $(x, g(x))$ [MP99]. Regarding Deep Learning terminology, g is called a task. In comparison to classical (linear) regression and model fitting, where representable functions are limited by a handcrafted choice of basis functions, neural networks gain their strength in stacking a large amount of simple, but yet highly parameterized function blocks, so that in theory even the most complex functions can be accurately approximated. Thus, neural networks were successfully deployed in a vast variety of difficult and as of yet poorly understood domains such as computer vision [Dai+17; Mil+20; Zol+18], rendering [Tew+20; Gar+21; Tak+21; TZN19], robotics [Kár+21; Mor+21], medical applications [Aza+21; Bha+21; KA21], light transport [Kal+17; Mül+21; Mül+22], and physical simulations [EUT19; Thu+20; Umm+19].

A feedforward neural network is a highly parameterized function f_θ with the parameters θ called network parameters, network weights, or simply weights. Given a vector-valued input x , the network transforms x to another scalar or vector-valued output $f_\theta(x)$ describing the solution of a task the network has been trained for. The structure of f_θ follows a set of simple, piecewise differentiable function blocks that are organized in a directed acyclic graph called computation graph. Each function block g computes an output by requiring some network weights and one or more input values x_1, x_2, \dots being either the task-related input x or intermediate values computed by other blocks. In the latter case, edges pointing to g are introduced in the computation graph to indicate its dependencies. The

full network f_θ is evaluated by doing a forward pass through the computation graph and invoking the function blocks in such an order that all dependencies are resolved.

5.1. Training

In order to fit f_θ to a specific task g , an optimization process modifying the weights θ , called training, is necessary. First, a task-specific loss function ℓ is selected. It measures the quality of the current network configuration by comparing a network output $f_\theta(x)$ to the expected outcome $g(x)$. Common loss functions are the mean absolute error (MAE) loss

$$\ell_{\text{MAE}}(x, y) = \frac{1}{n} \sum_i^n |x_i - y_i| \quad x, y \in \mathbb{R}^n,$$

the mean squared error (MSE) loss

$$\ell_{\text{MSE}}(x, y) = \frac{1}{n} \sum_i^n (x_i - y_i)^2 \quad x, y \in \mathbb{R}^n,$$

or—in the case of probabilistic and classification tasks—the cross-entropy loss

$$\ell_{\text{cross-entropy}}(p, q) = - \sum_i^n q_i \log p_i$$

where $p, q \in \mathbb{R}^n$ encode probabilities over n discrete events. Second, the function g has to be sampled as densely as possible to generate a large dataset \mathcal{D} of input/ground truth pairs $(x, g(x))$. For most real-world tasks, this step implies an extensive data acquisition process of generating and labeling data in the wild. For rendering applications such as adaptive super-sampling (see Chap. 6), data can be generated automatically by generating random scene configurations x and deriving $g(x)$ from high-resolution renderings. Ultimately, a first-order gradient descent optimization solves for

$$\arg \min_{\theta} L(\theta, \mathcal{D}) \quad \text{where} \quad L(\theta, \mathcal{X}) := \frac{1}{|\mathcal{X}|} \sum_{(x, g(x)) \in \mathcal{X}} \ell(f_\theta(x), g(x)) \quad (5.1)$$

by iteratively updating the network weights θ according to the update formula

$$\theta_{(n)} = \theta_{(n-1)} - \eta \nabla_{\theta} L(\theta_{(n-1)}, \mathcal{D}). \quad (5.2)$$

Here, η is the learning rate that controls the convergence of the optimization method. Oscillating or unstable updates in the case of highly non-linear objectives can be mitigated by choosing a low value for η . On the downside, this also increases the number of iterations until convergence is reached.

Backpropagation Each first-order optimizer requires access to the gradient of $\theta \mapsto \ell(f_\theta(x), g(x))$. Since function blocks, as well as loss functions used in Deep Learning, have closed-form derivatives, this can be achieved with the chain rule. It states that

$$\frac{\partial(v \circ u)}{\partial x}(x) = \frac{\partial v}{\partial x}(u(x)) \cdot \frac{\partial u}{\partial x}(x),$$

where $u: \mathbb{R}^\ell \rightarrow \mathbb{R}^m$ and $v: \mathbb{R}^m \rightarrow \mathbb{R}^n$ are two differentiable functions. The chain rule implies that if the subsequent gradient $\frac{\partial v}{\partial x}(u(x))$ is already known, then the gradient of the composition can be obtained by evaluating $\frac{\partial u}{\partial x}(x)$ on the input and computing a matrix product. In particular, gradients of the loss function w.r.t. any function block can be obtained by caching all function block inputs in the forward pass and then doing a backward pass through the whole computation graph. Starting with the loss function, the gradient $\frac{\partial \ell}{\partial x}(f_\theta(x), g(x))$ is evaluated and cached as outgoing gradient at the last node. Whenever a function block h of the network with cached outgoing gradient y and cached input x is reached, the partial derivatives of its inputs x_i are computed via $y \cdot \frac{\partial h}{\partial x_i}(x)$ and then are added to the gradient caches at the function blocks outputting the respective inputs. Similarly, weight gradients $y \cdot \frac{\partial h}{\partial \theta}(x)$ are computed and cached as well. After completing the backward pass, the gradient of $\theta \mapsto \ell(f_\theta(x), g(x))$ can be read off the weight gradient cache.

The whole process of traversing the computation graph from back to front while tracking gradients is called backpropagation [RHW86]. Since all function block inputs have to be cached, backpropagation has a significantly higher memory footprint compared to a simple forward pass without caching. For this reason, training a network requires notably more GPU memory than evaluating it. By caching only a subset of function block inputs at a time and recomputing others by a second forward pass through a small part of the computation graph, memory consumption can be decreased at the cost of additional forward pass computations. This technique is called gradient checkpointing [Che+16].

Stochastic Gradient Descent In practice, datasets are far too large to evaluate the network on every sample once per gradient descent step. To make the training process tractable, stochastic gradient descent (SGD) repeatedly draws minibatches $\mathcal{B} \subset \mathcal{D}$ without replacement and performs a gradient descent step on the objective $L(\theta, \mathcal{B})$. If minibatches are chosen sufficiently small, the objective and its gradient w.r.t. θ can be computed in a single forward-backward pass without running out of memory by caching function block inputs. Additionally, this approach allows escaping bad local minima of Eq. (5.1) by frequently changing the data samples for which the loss is computed.

Adaptive Moment Estimation Gradient Descent via the update rule Eq. (5.2) requires stable gradients $\nabla_\theta L$ throughout the training process in order to function reliably. Especially when computation graphs of networks become deeper, the quality of gradients starts to erode. Even more, depending

on the actual network architecture, gradients tend to diminish or explode when tracking them in the backpropagation pass. As a consequence, weights appearing at different levels of the computation graph exhibit a high dynamic range in their gradients. Whereas some weights remain mostly unchanged in Eq. (5.2), others change (too) drastically. A decent global learning rate η thus cannot be found.

Adaptive moment estimation (ADAM) by Kingma and Ba [KB14] is a highly successful first-order optimization scheme that addresses the aforementioned issues. It replaces the update scheme Eq. (5.2) by

$$\theta_{(n)} = \theta_{(n-1)} - \eta \frac{\widehat{m}_{(n)}}{\sqrt{\widehat{v}_{(n)} + \varepsilon}}, \quad (5.3)$$

where \widehat{m} , \widehat{v} are unbiased, exponential moving averages of the first and second order moments of $\nabla_{\theta} L$, and $\varepsilon = 10^{-8}$ is a small number to avoid a singularity at $\widehat{m} \approx \widehat{v} \approx 0$. The biased moment averages are updated via

$$\begin{aligned} m_{(n)} &= \beta_1 \cdot m_{(n-1)} + (1 - \beta_1) \cdot \nabla_{\theta} L(\theta_{(n-1)}, \mathcal{B}) && \text{where } m_{(0)} = 0 \\ v_{(n)} &= \beta_2 \cdot v_{(n-1)} + (1 - \beta_2) \cdot (\nabla_{\theta} L(\theta_{(n-1)}, \mathcal{B}))^2 && \text{where } v_{(0)} = 0 \end{aligned}$$

in each iteration, and then corrected by $\widehat{m}_{(n)} = m_{(n)}/(1 - \beta_1^n)$ and $\widehat{v}_{(n)} = v_{(n)}/(1 - \beta_2^n)$ to avoid a bias towards zero in the starting phase.

The new update scheme has two stabilizing properties. First, gradients are smoothed significantly by applying a very small amount of decay—Kingma and Ba suggest to use $1 - \beta_1 = 0.1$, $1 - \beta_2 = 0.001$ —and second, high variations in the gradient lead to smaller updates since the ratio m/\sqrt{v} decreases. Last but not least, Eq. (5.3) is scale-invariant, that is, the same update happens even if the gradient is multiplied by a fixed constant in each iteration step. For this reason, it is possible to find a learning rate parameter η that suits all trainable weights at once.

The discussion of ADAM concludes our introduction to training procedures. At this point, we note that even with ADAM, training is not yet trivial. We have only scratched the surface of related topics to provide an intuitive idea of how training proceeds in principle. We did not discuss over- & underfitting, adaptive learning rates, regularizations such as dropout [Sri+14], batch normalization [IS15] or weight decay, the issue of catastrophic forgetting, or data augmentation.

5.2. Neural Network Architectures

Next, we briefly introduce the function blocks and neural network architectures used in the works accompanying the thesis. More specifically, multilayer perceptrons (MLPs) as well as the principal architecture choices behind GoogLeNet [Sze+15], DenseNet [Hua+17] and SRNet [SVB18] are discussed. We only present a very small subset of neural network classes. In general, there exist

many more network architectures that we are not going to cover, for instance (variational) autoencoders [KW13; HZ93], generative adversarial networks (GANs) [Goo+14] and recurrent neural networks (RNNs) [Hop82] such as Transformer Networks [Vas+17] and long short-term memory (LSTM) networks [HS97].

Fully Connected Layer & Activation Functions The most commonly used function block used in Deep Learning consists of an affine transformation $y = Mx + b$ followed by an activation function $a(y)$ that exhibits some non-affine behavior. Given the input and output dimensions $n_{\text{in}}, n_{\text{out}}$, the affine transformation is fully defined by the linear transformation matrix $M \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ and the bias $b \in \mathbb{R}^{n_{\text{out}}}$. The entries of both M and b are part of the network weights θ and are fully adjustable in a subsequent training process. Note that every component of the output y depends on all components of x so that a dependency graph over the components of x and y would be a fully connected, bipartite graph. Hence, this special type of function block with arbitrarily parameterized M and b is called a fully connected layer.

Since affine transformations are closed under composition, any network utilizing only mappings $x \mapsto Mx + b$ is inevitably affine itself. To disrupt this symmetry, a non-affine activation function a is applied to each vector component after transformation. Hence, the outputs of a function block are called activations. In our works, we utilize the well known rectified linear unit (ReLU) given by $\text{ReLU}(x) := \max(0, x)$. Arguably, it may be the most simple choice for a non-affine transformation. It prevailed in literature due to the simple and fast evaluation of $\text{ReLU}(x)$ and $\partial \text{ReLU}(x)/\partial x$, so that more computational effort can be spent on other parts of the network. Depending on the actual type of signals one intends to process, other activation functions such as SIREN (periodic), leaky ReLU (non-vanishing gradients), or SoftPlus (smooth) can be utilized. We consider an extensive overview and discussion of activation functions to be out of scope for this thesis and refer to the survey of Apicella et al. [Api+21].

Convolutional Layers Convolutional neural networks (CNNs) [Fuk80] utilize function blocks describing convolutional operations. They are commonly used in applications processing 1D temporal, 2D image, or 3D volumetric data. In all of these scenarios, data samples are equipped with an additional temporal or spatial structure which suggests that close-by data samples are highly correlated. In particular, the presence—or even more importantly—the absence of certain correlations in local surroundings are of particular interest when reasoning about such data. For instance, it is well known that image processing neural networks develop edge filters that are sensitive to the lack of correlation between two adjacent regions [ZF14].

A 2D convolutional layer expects a tensor $x \in \mathbb{R}^{c_{\text{in}} \times h \times w}$ describing c_{in} 2D maps of size $h \times w$ and then convolves its with a fully trainable set of filter kernels $k \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times h \times w}$ such that the i -th slice of

$y \in \mathbb{R}^{c_{\text{out}} \times h \times w}$ is formed as the sum $\sum_j x_j * k_{i,j}$ where $*$ is the discrete 2D convolution operator. The final output of the layer is obtained by adding an optional, spatially independent bias vector $b \in \mathbb{R}^{c_{\text{out}}}$ to y and applying an activation function. Convolutional layers are equivariant w.r.t. translation and make use of local spatial information when deriving outputs. Both properties are highly desirable in applications as mentioned above. Note that after flattening x and y into a single dimension, the convolution operator can be considered a special case of an affine transformation where most of the matrix M is forced to zero and many entries in M and b share the same weights. A transposed convolutional layer is a function block where the matrix M is defined by a set of filter kernels as above, but then $M^T x + b$ is computed instead. The concept of 2D convolutional layers translates directly to arbitrary dimensions, although it has to be mentioned that already for three dimensions convolutional layers have a critically high memory footprint so that only small input resolutions are feasible [MS15].

In the following, we describe the networks architectures we have deployed. Regarding learning generic shape properties [RW22] (see Chap. 6), we use MLPs to en- & decode geometry patches. Further, we have tested inverse feature visualization [RW20] (see Chap. 7), on the Inception network GoogLeNet [Sze+15], the DenseNet-BC ($k = 12$) incarnation and SRNet.

Multilayer Perceptron MLPs are realized by stacking an arbitrary amount of fully connected layers. That is, the function f of the neural network is given by $f = f_d \circ f_{d-1} \circ \dots \circ f_1$ where each function block f_i is of the form $a(Mx + b)$, with a being the activation function. Due to their simplistic design, MLPs are mostly deployed in applications where a) the input vector x has low dimension and/or its components are unrelated to each other, and b) forward passes have to adhere to tight timing constraints.

Recently, MLP architecture became popular again for scene reconstruction and rendering tasks. Here, networks have to be invoked ten to a hundred million times per second to achieve real-time performance with per-pixel network invocations. Recent works of Müller et al. [Mül+21] and Weiss et al. [WHW21] show that such performance levels are realizable on modern commodity hardware by providing custom CUDA implementations of small-scale MLPs exploiting NVIDIA Tensor Cores.

Inception Networks The Inception network architecture as proposed by Szegedy et al. in 2015 [Sze+15] is a special class of CNNs that was successfully utilized for classifying images with state-of-the-art performance at that time. Inception blocks consist of multiple convolutional layers that are connected in parallel. By varying the filter kernel sizes for each layer, the network can pick the filter radius that performs best for the given task. An Inception network is realized by stacking Inception modules and occasionally weaving in a max-pooling layer that reduces the spatial resolution of activations by applying max operations over 2×2 regions.

DenseNet Huang et al. presented the CNN-based DenseNet architecture in 2017 [Hua+17]. DenseNets are created by stacking a small number of dense blocks and transition layers that consist of a convolutional and a pooling layer to reduce the spatial resolution. In comparison to other designs where function blocks are chained one after another, dense blocks consist of multiple convolutional layers that take in not only the output of the previous layer but the outputs of all preceding layers arising in the current block. Thus, DenseNets encourage the reuse of internal activations over multiple levels, allowing for network designs with overall fewer network weights compared to previous CNNs. Further, backpropagation paths are shortened so that problems in training deep architectures with many layers (see Sec. 5.1) are mitigated.

SRNet SRNet is a fully convolutional neural network architecture that is used for upsampling in video super-resolution [SVB18]. Besides using transposed convolutional layers over strided kernels for upsampling, it also incorporates multiple residual blocks with two convolutional layers each. A residual block requires identical in- and output formats so that the input can directly be added to the output of the block via a residual connection [He+16]. In doing so, the network is biased toward learning the identity mapping. Similar to DenseNet, residual connections enable the training of deeper network architectures.

Learning Generic Local Shape Properties for Adaptive Super-Sampling

Voxel-based geometry representations can encode volumetric as well as surface models in a simple yet compact fashion that can be efficiently rendered via ray-tracing. The central data structure, the sparse voxel octree (SVO), natively supports a simple level of detail (LOD) scheme, and it allows to place material information at the same resolution level as the represented geometry. In contrast, LOD hierarchies for mesh-based representations have to be carefully designed [DSW09; SW06] to avoid holes at LOD transitions, to prevent geometry popping, and to be memory efficient by reusing as much information as possible from coarse levels when describing the next finer geometry level. Providing materials with a mesh-based surface requires uv-texture mapping, a technique that adds additional complexity by requiring a 2D surface parametrization. For most geometric models, finding such a parametrization is a non-trivial task since distortions have to be considered and seams must be placed. Voxel-based texturing can avoid uv-texture mapping completely [DSA20].

As a downside, SVOs disregard partial occlusions of local geometry in screen space, so that the view-independent, material information stored at coarse SVOs nodes may be highly misleading. In comparison, LOD texture representations for meshes such as mipmaps [Wil83] suffer from view-dependent screen space distortion effects. However, errors introduced by distortions are less striking and can be accurately addressed through anisotropic filtering [Ewi+00]. In our short paper, we address the aforementioned disadvantage of SVOs by providing an adaptive super-sampling approach that detects and refines errors due to LOD voxel rendering [RW22]. Our main contribution lies in applying a learned, compact shape representation to a rendering task and proving its effectiveness.

In the following, the basics of LOD-based voxel rendering and its failure cases are presented in Sec. 6.1. Then, we present the PatchNet architecture in Sec. 6.2 and explain in Sec. 6.3 how it can be utilized to derive super-sampling patterns.

6.1. Sparse Voxel Octrees & Level of Detail

Voxel geometry is represented by storing occupancy information in a high-resolution 3D voxel grid. To even capture small geometric details in large scenes, grid resolutions of $2K^3$ and higher are required. For instance, to accurately resolve maple leaf shapes in the San Miguel scene (see our paper), we utilize a grid resolution of $8K^3$. Clearly, even storing a dense bit mask over an $8K^3$ grid is intractable, requiring 64GB of data. An SVO limits memory consumption by organizing the grid hierarchically in an octree where subtrees not containing any geometry are pruned from the representation. Each node in the SVO contains an 8bit child mask, indicating which subtrees are populated, as well as a pointer to each of the existing children. At the leaf level, shading information such as voxel color is stored. Hence, any grid of resolution $(2^n)^3$ can be represented by an SVO with $n + 1$ levels.

To render the voxel geometry w.r.t. a specific viewport, one ray per pixel is shot through the pixel’s center and then intersected with the SVO. First, the ray is tested against the root node. If an intersection with the node’s corresponding voxel occurs, testing is recursed by checking for intersections with the children of the root node in the order as the ray would pass them. The procedure stops at the first intersection with a leaf node voxel, where the stored shading information is used to determine the final pixel color. If the ray exits the root node without intersecting any leaf, the pixel is colored according to the scene’s background. LOD rendering is realized by equipping internal SVO nodes with averaged shading information and stopping ray-SVO traversal early at the first intersection—not necessarily at leaf level—for which the size of the intersected voxel matches the size of the pixel cone associated to the ray. Since averaged shading information is used to shade a pixel, undersampling artifacts such as aliasing or temporal flickering during animations are mitigated.

Failure Cases When rendering a pixel with averaged shading information taken from an internal SVO node, it is assumed that the leaf voxels in the respective SVO subtree fully cover the pixel and contribute in equal parts. This assumption is violated if the geometry represented by the intersected voxel suffers from severe self-occlusion for the current view, or projects only to a small portion of the pixel. For instance, in terrain rendering, mountain peaks that are lit from only one side during dawn or dusk will be perceived in a grayish tone at low levels of detail, regardless of the viewer’s perspective. Here, dark colors from the unlit side are erroneously blended with the bright colors of the lit side. The LOD representation does not account for the fact that half of the mountain is occluded from most views. When distance is reduced, color-popping artifacts occur as soon as the rendering switches to a higher LOD in which both sides of the mountain peak are separated. A similar effect can be observed for stacks of close-by, thin geometry sheets (see Fig. 6.1 top). When viewing fence-like structures (see Fig. 6.1 bottom), part of the background can be perceived through the laths. At low levels of detail, gaps between laths are artificially closed, so that any contribution from the background is missed out.

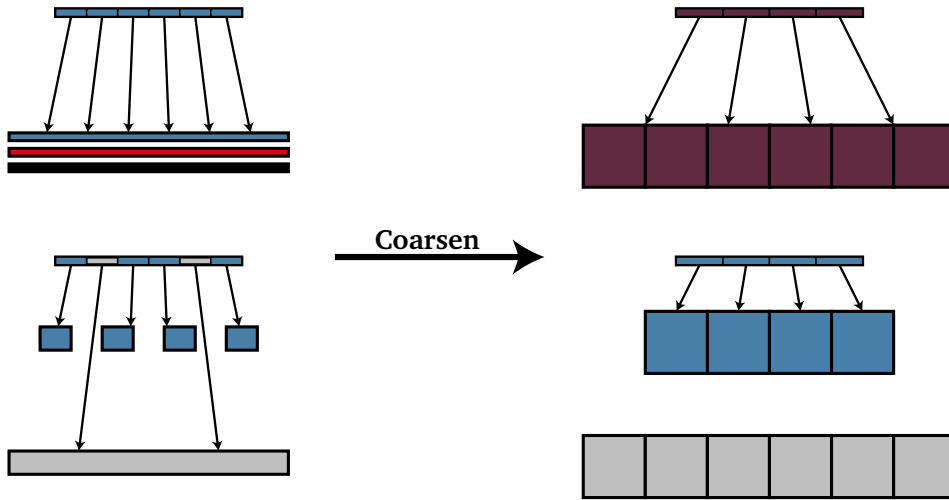


Figure 6.1.: Failure cases of voxel LOD representations. The left side shows actual voxel geometry, whereas the right side depicts its coarse representation. Note that errors in the low LOD rendering result in a uniformly shaded viewport so that screen-space-based super-sampling will not issue more samples. Figure has been adapted from our presentation held at EuroGraphics 2022.

In our work, we propose a method to detect such cases. By sampling the geometry at a higher LOD with an increased ray-sample count, errors can be corrected. Note that existing adaptive super-sampling approach operating in screen-space [LRU85; Mit87; RFS03a; RFS03b; Xu+07] fail to correct errors, since neighboring pixels show similar, yet consistently wrong appearance in the scenes of Fig. 6.1.

6.2. PatchNet

We argue that screen-space approaches do not attack the issue with LOD rendering at its core, namely occluding geometry. Instead, we propose a world-space approach that considers local geometry at ray-voxel intersections when reasoning about super-sampling. It builds upon a neural network f that takes a ray-voxel intersection point p , as well as additional rendering parameters θ that describe the projection to the shaded pixel. The network returns an occupancy value σ that estimates the fraction of the pixel covered by local geometry around p , and it also returns the average luminance L projected into this fraction. Our super-sampling approach now operates as follows: For each ray intersecting with the SVO, we evaluate f and check if the returned occupancy is close to one, and if the returned luminance is close to the luminance stored in the intersected voxel. If either condition is not satisfied, super-sampling is issued. We realize f by training an encoder/decoder network that we call PatchNet. Its architecture is shown in Fig. 6.2. The encoder takes in local geometry patches from

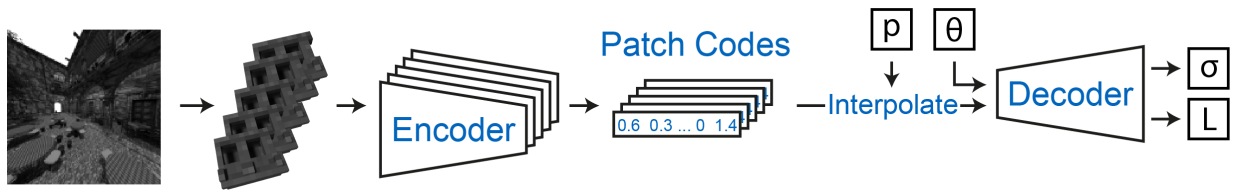


Figure 6.2.: Overview of PatchNet. Given a scene represented by an SVO, its local geometry patches are encoded in low-dimensional feature vectors called patch codes. Given a ray-voxel intersection p , its surrounding patch codes are interpolated to derive an abstract descriptor of the local geometry surrounding p . Together with the view-dependent rendering parameters θ , the subsequent decoder interprets the interpolated patch code and derives occupancy and luminance. Figure has been taken from our presentation held at EuroGraphics 2022.

the current scene and transforms them into latent feature vectors—so-called patch codes. Together with the view-dependent rendering parameters, the subsequent decoder acts as a lightweight neural renderer. It interprets the patch codes and derives view-dependent occupancy and luminance values. We refer to Sec. 2.3 of our paper for a detailed description of the training process.

Our design comes with several advantages: First, PatchNet does not depend on a specific scene; a dependency which otherwise is common for geometry encoding networks such as scene representation networks [SZW19]. By depending on local geometry information only, PatchNet immediately generalizes to arbitrary scenes that are composed of similar geometry patches seen in training. As observed in our work as well as other works by Jiang et al. [Jia+20] and Liu et al. [Liu+20a], the space of plausible, local geometry shapes is rather limited. It can be encoded in a low-dimensional feature vector. In our experiments, it suffices to train ten-dimensional patch codes to solve the super-sampling task and generalize to other datasets. As a second advantage, the encoder/decoder structure of PatchNet permits the computation of all scene-specific patch codes by executing the encoder of the network in a preprocess. During real-time rendering, only the lightweight decoder has to be evaluated per pixel, thus relieving a large chunk of the computational burden introduced by the overall architecture.

Geometry Encoding The encoder of PatchNet takes responsibility for transforming the scene into a hierarchical representation of patch codes that follow the same structure as the scene SVO. Inspired by the work of Takikawa et al. [Tak+21], we store a patch code per corner of each occupied, internal voxel of the tree. By trilinear interpolation, one obtains a continuous multi-level hierarchy of geometry descriptors that can be sampled from the SVO at any ray-voxel intersection at any LOD. We also tested if trilinear interpolation can be replaced by inputting interpolation weights and patch codes of all corners directly to the decoder. In doing so, one can further reduce the number of stored patch code components per voxel corner to two, since the decoder still receives a 16-dimensional patch code by

concatenating at eight corners. However, this design choice comes at the cost of PatchNet losing some of its ability to generalize to other datasets. Hence, we did not present it in the paper.

The patch code at a voxel corner c at LOD ℓ is derived as follows. First, we sample a vector-valued representation of the geometry surrounding the corner. The eight voxels touching c span a cubical region R of side length $2^{1-\ell}$. Due to early termination in LOD-enabled ray-SVO traversal, it is guaranteed that for any ray intersecting the SVO at a point close to c at LOD ℓ , the pixel cone at the intersection point has at most a side length of $2^{-\ell}$. In particular, since the voxel corner c is located in the center of the pixel cone, and the maximal diameter of the pixel cone is $\sqrt{2} \cdot 2^{-\ell} < 2^{1-\ell}$, it is guaranteed that R covers the pixel completely. Thus, it suffices to pass the geometry information located at R to the encoder. We obtain it by querying the SVO at level $\ell + \Delta\ell$ and populating a dense 3D grid of resolution $(2^{1+\Delta\ell})^3$ containing occupancy and shading data. Then, the grid is flattened and passed to the encoder where it is processed by three fully connected layers of 256 activations each and one terminal fully connected layer that outputs the ten-dimensional patch code. Since the encoder operates on local geometry information only, it is not required that geometry passed to the encoder has to originate from the same scene the encoder was trained on.

Decoder The decoder is realized by a small-scale MLP with three fully connected layers outputting 48-dimensional activations each, and a final fully connected layer outputting occupancy and luminance. To obtain optimal approximations of the ground truth occupancy and luminance, we experimented with various inputs that can be passed to the decoder. Combinations of the following input types were considered:

- Ray direction in cartesian coordinates.
- Ray direction encoded as spherical harmonics. That is, the view direction is evaluated on the $(d + 1)^2$ spherical base functions up to degree d , and then the results are passed to the decoder.
- Up vector of the camera projected to the orthogonal complement plane of the current ray direction.
- Pixel-voxel-ratio encoding the relative size of the projected voxel to the current pixel. It is obtained by dividing the side length of the pixel by the side length of the intersected voxel after projecting it onto the screen.
- (Averaged) surface normal in world space.
- Surface normal in world space, encoded as spherical harmonics.
- Surface normal in camera space.
- Surface normal in camera space, encoded as spherical harmonics.

- Angle φ between the surface normal and view direction, encoded via the Fourier series. That is, the values $\cos(n\varphi)$ and $\sin(n\varphi)$ are provided as input up to a fixed upper degree of $n = 8$.
- Roll of the camera w.r.t. the up vector $(0, 1, 0)$, again encoded via the Fourier series.

Additionally, we tested various patch code dimensions and multiplied some of the patch code components with various combinations of

- Spherical harmonics encoding of the view direction up to degree 2, thus multiplying up to 9 patch code components.
- Spherical harmonics encoding of the surface normal in world space up to degree 2.
- Spherical harmonics encoding of the surface normal in camera space up to degree 2.
- Fourier series terms of the roll of the camera up to $n = 6$, thus multiplying up to 12 patch code components.
- Fourier series terms of the angle between the surface normal and view direction up to $n = 6$.

By multiplication with spherical harmonics or Fourier base functions, patch code components are effectively reinterpreted as spherical harmonics (or Fourier) coefficients. After an extensive ablation study considering all combinations from the two lists above, we settled on the specific input format as described in our paper, see [RW22, Sec. 2.2]. It yields close to optimal performance for input of moderate dimensionality. Similarly, we tested different MLP designs by varying the number of fully connected layers as well the number of activations they output. We found that our decoder design yields good results while still being small enough so that it can be evaluated in shared GPU memory as proposed by Weiss et al. [WHW21]

6.3. Derivation of the Super-Sampling Pattern

To determine which pixels require super-sampling, we use PatchNet to implement a scoring function that is invoked per pixel. If a pixel’s score supersedes a user-defined threshold, it will be refined. Instead of providing a score threshold, one can also define a fixed budget of X pixels. Then, the X top-scoring pixels are marked for super-sampling. Let σ_N, L_N be the occupancy and luminance output of the network. Further, let L_V be the average luminance stored at the intersected voxel of the SVO. We considered the following scoring functions

- $|1 - \sigma_N| + \lambda|L_V - L_N|$ where $\lambda \in \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$
- $|1 - \sigma_N| + \sigma_N|L_V - L_N|$
- $|1 - \sigma_N|$
- $|L_V - L_N|$

and found that scoring with $|1 - \sigma_N| + 5|L_V - L_N|$ as presented in the paper yields the most effective super-sampling patterns.

Inverting the Feature Visualization Process for Feedforward Neural Networks

In the field of explainable artificial intelligence (XAI), feature visualizations are inputs to a neural network that stimulate a specific neuron [NYC19]. For networks in image classification, feature visualizations were used to present a large set of visual stimuli to a user, depicting entities that the network looks out for when classifying images [Car+19]. By inspecting a dashboard of such visualizations [Ola+18; Car+19], users can identify the various concepts learned by the network. Even more, these dashboards also provide a means to compare networks with each other by searching for similar visual stimuli occurring in multiple networks at once.

Before feature visualizations became popular, similar neurons were identified by automatically searching for correlations in their response behavior when being confronted with an existing dataset of input images [ZF14]. However, if a neuron looks out for features that are not present in the dataset, it will not generate any significant activations and therefore is overseen. As a consequence, any approach to analyzing neuron activations always is biased toward a specific dataset. In the worst case, analysis is performed over a dataset that does not reflect the input domain the network was trained for. Then, many important correlations will not emerge, and others may be misinterpreted.

Feature visualizations, on the other hand, do not depend on a specific dataset. They are generated by a process called activation maximization (AM) [Erh+09]. Given a network N and a network input x , AM generates a stimulus x^* of neuron i by optimizing for

$$\arg \max_x N(x)_i, \quad (7.1)$$

where $N(x)_i$ describes the activation of neuron i when providing x as input to the network. Then, x^* visualizes what a neuron “would like to see”. For instance, if x^* shows dog snout like shapes, it is very likely that neuron i is important for classifying dogs.



Figure 7.1.: Exemplary feature visualizations obtained when applying regularized activation maximization. Figure from “Feature Visualization” by Olah et al. [OMS17], licensed under Creative Commons Attribution CC-BY 4.0.

In general, feature visualizations often show highly abstract drawings of competing concepts, and their interpretation requires some creativity and experience, see Fig. 7.1. We highly suggest browsing through the appendix material provided by Olah et al. [OMS17] to develop a notion of what stimuli may look like for GoogLeNet. Although it is claimed that analogous features form across models and tasks [Ola+20], to our knowledge, feature visualizations were not used yet in any downstream algorithm that can automatically match neurons of similar characteristics. Instead, features visualizations are integrated in multiple XAI tools [Car+19; Goh+21; Hoh+20; OMS17; Ola+18] to communicate semantic meanings of neurons to the user. Interpretation and matching of visualizations remain to be done by the user. We intend to address this issue by proposing inverse feature visualization (IFV) in our work [RW20]. That is, given x^* , we intend to find the most likely neuron i —or a weighted linear combination of neurons i_j with weights w_j —such that x^* would be returned by optimizing for

$$\arg \max_x \sum_j w_j N(x)_{i_j}. \quad (7.2)$$

The vision behind IFV is as follows: The user (or algorithm) selects and visualizes a neuron using network A first, and then inverts the resulting visualization with IFV using network B. This yields two neurons obeying the same visualization and, thus, representing the same semantic concept. One can utilize this technique in a feature-based comparison of networks to generate insights into the relevance of patterns for successful network training. In our work, we propose gradient-based inverse feature visualization (Grad-IFV) to realize IFV for a single network. Given a linear combination of neurons of network A, we can fully recover them just from knowing the corresponding feature visualization obtained by optimizing Eq. (7.2) regarding A.

However, we did not succeed in bringing the full vision behind IFV to life. As soon as the visualization obtained from network A is inverted with regard to network B, results are unsatisfactory, even if both networks were trained on the same task and the investigated neuron resides in the classification layer of A. It was expected to at least be able to match neurons in the final classification layer. We assume that IFV fails since, in real scenarios, a visualization that is optimal for one network will not be so for another network. It might not even be close to an optimum as long as the sensitivity of the objective function Eq. (7.2) to high frequencies and noise is not reduced.

At that point, we accepted the inapplicability of our approach to real-world tasks and decided to cease further attempts in publishing our results. Nonetheless, in student work, we further pursued the idea of IFV and integrated concepts such as image priors or transformation robustness [OMS17] into our work. Further, being in line with the insights of Roberts and Tsiligkaridis [RT21], we hypothesize that feature analysis methods such as IFV can yield consistent results if networks under consideration were trained by a robust training approach, counteracting high-frequency adversarial examples [GSS14]. As of yet, we did not investigate the behavior Grad-IFV on a robustly trained network and suggest revisiting Grad-IFV in this matter. Hence, we decided to present the implementation details of Grad-IFV in this thesis. After setting up the formal definition of IFV in Sec. 7.1, details and design decisions behind Grad-IFV are presented step by step in Sec. 7.2.

7.1. Inverse Feature Visualization

We repeat the definition of AM as given in our paper [RW20]. Let \mathcal{I} be the input domain of a network of interest, that is the set of all valid inputs with respect to the task the network has been trained for. Similarly, let N be a function such that $N(I)$ is a n -dimensional vector of all activations of the network when providing $I \in \mathcal{X}$ as input. Given a unit vector $x \in \mathbb{R}^n$ called target objective, AM optimizes for

$$\mathbf{I}^* = \arg \max_{I \in \mathcal{I}} S_x(N(I)), \quad (7.3)$$

with \mathbf{I}^* being called the realization of x . Here,

$$S_x(y) = x^T y \cdot (x^T y / \|y\|)^k \quad (7.4)$$

with $k \in \mathbb{N}_0$ is a measure of significance of the network activations y regarding x . Maximizing for a single neuron can be achieved by setting $x = e_i$.

Since AM neither has to be deterministic nor injective—i.e. different values for x do not necessarily infer different optima \mathbf{I}^* —a rigorous definition of IFV is more complex. For this reason, we consider AM as a random process Y over the domain \mathcal{I} with its probability density function h_Y depending on

x as additional parameter. Then, IFV means to compute the maximum likelihood estimator of Y , i.e. $\hat{x} = \arg \max_x \widehat{h}_Y(I; \mathbf{x})$ for a given input configuration I .

7.2. Gradient-Based Inverse Feature Visualization

Since realizations \mathbf{I}^* returned by AM are locally optimal solutions of the objective function

$$f_x : I \mapsto S_x(N(I)),$$

the necessary condition for optimality $\|\nabla f_x(I)\| = 0$ is supposed to hold. In order to recover the (most likely) target objective of a realization \mathbf{I}^* , we solve for

$$\arg \min_{\mathbf{x} \in S^n} \|\nabla f_x(\mathbf{I}^*)\|^2. \quad (7.5)$$

with S^n being the sphere of n -dimensional unit vectors. We call this approach gradient-based inverse feature visualization (Grad-IFV). As described in the paper, solving Eq. (7.5) directly fails due to trivial solutions for x . For instance, choosing any x with $x^T N(\mathbf{I}^*) = 0$ will result in either a saddle or minimum of f_x . In particular, its gradient vanishes, although x certainly is not the target objective we were ultimately interested in. We now present the various problems coming with optimizing Eq. (7.5) and present solutions to them.

Vanishing Gradients in the Significance Measure The chain rule states that

$$\nabla f_x(\mathbf{I}^*) = \nabla S_x(N(\mathbf{I}^*)) \cdot \nabla N(\mathbf{I}^*).$$

We show that $\nabla S_x(N(\mathbf{I}^*))$ is of the form $c \cdot x^T \cdot Z$ where c is a scalar depending on x and $N(\mathbf{I}^*)$, and $Z \in \mathbb{R}^{n \times n}$ is a symmetric, invertible matrix obtained by applying a rank one update of $N(\mathbf{I}^*)$ to the identity matrix. The scalar c vanishes if and only if $x^T N(\mathbf{I}^*) = 0$. As stated above, in this case one has found x such that \mathbf{I}^* is a saddle of f_x if k is odd, and a minimum otherwise. However, AM maximizes f_x and will not stop at either saddles or minima. Hence, we can safely drop c in the following reformulation of Eq. (7.5):

$$\arg \min_{\mathbf{x} \in S^n} \|x^T Z \nabla N(\mathbf{I}^*)\|^2. \quad (7.6)$$

At this point, the solution of Eq. (7.6) is given (up to a sign) by the left-singular vector to the smallest singular value of $M := Z \nabla N(\mathbf{I}^*)$. It is computed by first running a single forward and n backward

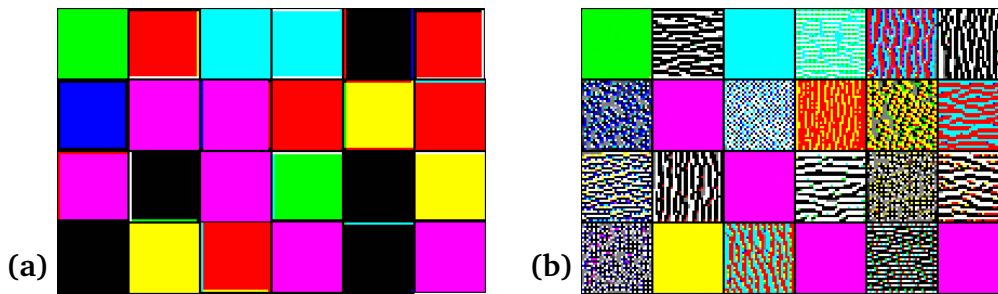


Figure 7.2.: Feature visualizations obtained when applying activation maximization (AM) to the feature maps of the first layer of a CNN. **(a)** Activations are summed up per feature map and then are jointly maximized. **(b)** A 3×3 max-pooling operation is applied to each feature map before continuing as in (a). Thus, on average, only every ninth activation contributes to the objective of AM. Note that edge- and stripe-like structures become visible now.

passes through the network f to determine $N(\mathbf{I}^*)$ and $\nabla N(\mathbf{I}^*)$. Then, $MM^T \in \mathbb{R}^{n \times n}$ is computed. The eigenvalue decomposition of MM^T yields the desired left-singular vector of M .

Reducing the Number of Neurons For state-of-the-art networks, the number of neurons n is in the millions so that the aforementioned strategy of solving Eq. (7.6) is intractable. In practice, however, feature visualizations act on specific layers of a network. Hence, we modify the function N such that it only returns neural activations of a preselected layer. If this layer is fully connected, n commonly reduces to numbers smaller than 1000. If the layer is a 2D convolutional layer instead, neurons form a 3D grid of c stacked, 2D feature maps with resolution $h \times w$. For instance, the inception-4a layer of GoogLeNet [Sze+15] outputs 512 feature maps of resolution 14^2 , a total of 100.352 neurons. However, each feature map of 14^2 neurons is derived from a single filter kernel (cmp. Sec. 5.2). Thus, all neurons of a single feature map are sensitive to the same features, but they search for them at different locations of the image. Olah et al. [OMS17] propose to pick either a single neuron per feature map or to sum up all activations of a feature map before applying AM. In both cases, the number of considered activations reduces to the number c of feature maps. Again, typically one has $c < 1000$, making our approach tractable.

Nonetheless, summing up activations along the spatial dimensions of a feature map has a significant catch that seems to have been overlooked by Olah et al. Consider a filter kernel w with weights $w_{i,j}$. Filtering an image I with pixels $I_{i,j}$ by convolving with w , and then summing up all pixels afterward, yields a linear transformation $I \mapsto \sum_{i,j} c_{i,j} I_{i,j}$ that linearly maps pixels in I to a scalar. However, for non-border pixels, the coefficient $c_{i,j}$ is given by summing up all filter weights $\sum_{i,j} w_{i,j}$. In particular, all information about the spatial structures detected by w is lost in the final signal and thus does not appear in feature visualizations. This issue becomes obvious when trying to generate feature visualizations at the first layer of a CNN. Although it is common knowledge that CNNs learn color

and edge detection in their first layer [ZF14], corresponding feature visualizations have—except for border regions—uniform colors, see Fig. 7.2a. We propose to break linearity by applying a max-pooling operation before summing over a feature map. In doing so, we obtain results that visualize edge detector features correctly, see Fig. 7.2b.

Bounded Input Domains More often than not, the input domain \mathcal{I} is restricted by some constraints. For instance, in image classification, each input image can only have RGB pixel values ranging from zero to one. Further, SRNet [SVB18] expects G-buffer information such as screen-space normal maps. Hence, some inputs are required to be unit vectors pointing into the Z -direction. In AM, such constraints can be considered by optimizing via projected gradient descent. In each step of projected gradient descent, the current solution for Eq. (7.3) is naively updated by ignoring all constraints and then reprojected to the constraint boundary $\partial\mathcal{I}$ if necessary. As soon as no further progress is achieved—for instance, because the current gradient is perpendicular to $\partial\mathcal{I}$ —optimization stops. Unfortunately, now the necessary condition for optimality, which is crucial to our approach, can be violated at $\partial\mathcal{I}$. This issue can be circumvented by means of a differentiable, surjective mapping $P: \mathbb{R}^{n_p} \rightarrow \mathcal{I}$ which parameterizes the input domain \mathcal{I} over a real-valued vector space of dimension n_p . If such a P is available, the constrained optimization problem Eq. (7.3) can be transformed to an unconstrained one over n_p variables via

$$\arg \min_{v \in \mathbb{R}^{n_p}} S_x(N(P(v))).$$

By replacing N with $N \circ P$, we can assume w.l.o.g. that \mathcal{I} is unbounded.

Linear Relationships in Network Gradients During our experiments, we noticed that the rows in the Jacobian $\nabla N(I)$ of the network exhibit linear relationships that persist for any choice of I . That is, the cokernel

$$\text{coker } \nabla N(I) := \{v \in \mathbb{R}^{n_f} \mid v^T \nabla N(I) = 0\}$$

of $\nabla N(I)$ is non-trivial. For instance, a dead neuron i absorbs all gradients and forces the i -th row of $\nabla N(I)$ to zero. As a consequence, $Z^{-1}e_i$ will always be a trivial solution to Eq. (7.6), independently of the realization \mathbf{I}^* and its target objective. Similarly, we noticed that gradients at the final layer of classification networks consistently cancel out when adding them up, i.e. $(1, \dots, 1) \in \text{coker } \nabla N(I)$ for any input I . Presumably, $(1, \dots, 1)$ arises as a trivial solution because the network learned to exploit the translation invariance of the softmax activation function at the last layer such that it can improve upon a regularization term on its weights.

Instead of solving Eq. (7.6), we propose to solve

$$\arg \min_{x \in \mathcal{S}^{n_f}} \|x^T Z \nabla N(\mathbf{I}^*)\|^2 \quad \text{s.t.} \quad Zx \in C, \quad (7.7)$$

where the critical space

$$C := \left(\bigcap_{I \in \mathcal{I}} \text{coker} \nabla N(I) \right)^\perp \quad (7.8)$$

masks out solutions which are valid with regard to all inputs in \mathcal{I} . In practice, we compute C as follows: First, we randomly sample inputs I and approximate the co-kernels $\text{coker} \nabla N(I)$ by computing the subspace of left-singular vectors of $\nabla N(I)$ with reasonable small singular values smaller than $\rho \cdot \|\nabla N(I)\|_2$. Here, ρ acts as a threshold that can be controlled via a nested intervals technique to determine all values of $\rho \in (0, 1)$ that yield critical spaces of different dimensions. Second, we deploy a relaxed scheme for computing intersections of subspaces which is based on interpolating principal vectors. Given two vector spaces U, V , their principal vectors [KA02] are recursively defined for $k = 1, \dots, \min(\dim U, \dim V)$ by

$$u_k, v_k = \arg \max_{u \in U, v \in V} u^T v$$

subject to

$$\|u\| = \|v\| = 1, \quad u^T u_i = 0, \quad v^T v_i = 0, \quad i = 1, \dots, k-1.$$

Even when U and V do not intersect, the principal vectors define the subspaces of U and V that are closest to each other in the sense of measuring principal angles $\theta_i = \arccos(u_i^T v_i)$. It holds that $\theta_1 < \theta_2 < \dots$, so that the k -dimensional subspaces in U, V being closest to each other are given by $\text{span}\{u_1, \dots, u_k\}$ and $\text{span}\{v_1, \dots, v_k\}$. In particular, by interpolating the generator pairs (u_i, v_i) , one can determine a subspace that is equally close to both U and V . By defining a threshold on the maximal allowed angle θ_k , one can thus find a vector space that approximates the intersection of $U \cap V$. Note that if the threshold is set to 0, the approximation becomes exact. However, in numerical computations U and V will most certainly never intersect. That is why we propose to approximate intersections with a rather generous threshold of 45° . Subspaces thus may be unnecessarily high-dimensional, but this is compensated for by intersecting up to 300 co-kernels for various inputs.

After determining the critical space, Eq. (7.7) can be transformed to an equivalent, unconstrained optimization problem over $x \in \mathcal{S}^{\dim U}$ that can be solved analytically with the same approach as described for Eq. (7.6). Please refer to our paper for details. There, we also show in an extended results section that Grad-IFV can invert the feature visualization process for a single network.

A. Parameterized Splitting of Summed Volume Tables

Abstract of Paper Summed Volume Tables (SVTs) allow one to compute integrals over the data values in any cubical area of a three-dimensional orthogonal grid in constant time, and they are especially interesting for building spatial search structures for sparse volumes. However, SVTs become extremely memory consuming due to the large values they need to store; for a dataset of n values an SVT requires $\mathcal{O}(n \log n)$ bits. The 3D Fenwick tree allows recovering the integral values in $\mathcal{O}(\log^3 n)$ time, at a memory consumption of $\mathcal{O}(n)$ bits. We propose an algorithm that generates SVT representations that can flexibly trade speed for memory: From similar characteristics as SVTs, over equal memory consumption as 3D Fenwick trees at significantly lower computational complexity, to even further reduced memory consumption at the cost of raising computational complexity. For a $641 \times 9601 \times 9601$ binary dataset, the algorithm can generate an SVT representation that requires 27.0GB and 46.8 data fetch operations to retrieve an integral value, compared to 27.5GB and 1521.8 fetches by 3D Fenwick trees, a decrease in fetches of 97%. A full SVT requires 247.6GB and 8 fetches per integral value. We present a novel hierarchical approach to compute and store intermediate prefix sums of SVTs, so that any prescribed memory consumption between $\mathcal{O}(n)$ bits and $\mathcal{O}(n \log n)$ bits is achieved. We evaluate the performance of the proposed algorithm in a number of examples considering large volume data, and we perform comparisons to existing alternatives.

Author Contributions The first author is responsible for the overall idea, implementation as well as all studies presented in this work. The paper was composed by the first author under consideration of feedback as well as minor revisions by Prof. Dr. Rüdiger Westermann.

Note A presentation held by the first author is publicly available at

<https://youtu.be/JSHjLvIu1Y0?t=1392>

Copyright © 2021 The Author(s). Computer Graphics Forum published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd. Reprinted under the Creative Commons CC-BY-NC license with permission from Rüdiger Westermann.

B. Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques

Abstract of Paper We use k -order Voronoi diagrams to assess the stability of k -neighborhoods in ensembles of 2D point sets, and apply it to analyze the robustness of a dimensionality reduction technique to variations in its input configurations. To measure the stability of k -neighborhoods over the ensemble, we use cells in the k -order Voronoi diagrams, and consider the smallest coverings of corresponding points in all point sets to identify coherent point-subsets with similar neighborhood relations. We further introduce a pairwise similarity measure for point sets, which is used to select a subset of representative ensemble members via the PageRank algorithm as an indicator of an individual member's value. The stability information is embedded into the k -order Voronoi diagrams of the representative ensemble members to emphasize coherent point-subsets and simultaneously indicate how stable they lie together in all point sets. We use the proposed technique for visualizing the robustness of t-SNE and multi-dimensional scaling applied to high-dimensional data in neural network layers and multi-parameter cloud simulations.

Author Contributions The overall idea of the paper was developed by the first author in discussions with Prof. Dr. Rüdiger Westermann. The first author is responsible for all implementations except for the glyph realization in Member-centric cluster variability plots, which is credited to Dr. Alexander Kumpf. The first author conducted all studies not related to Member-centric variability plots. This includes performance tests, the choice of the order parameter (Sec. 5.1), the sensitivity of Voronoi plots (Sec. 5.6) as well as the verification of representative projections (Sec. 5.7). The remaining studies were designed, performed, and discussed by the first author and Dr. Alexander Kumpf with equal contributions. The paper was composed by the first author, with extensive revision of introduction, related work, and conclusion by Prof. Dr. Rüdiger Westermann. Dr. Alexander Kumpf contributed Sec. 5.2 and related work regarding ensemble visualization and clustering.

Note A presentation held by the first author is publicly available at

<https://youtu.be/gbk68z0fVCE?t=229>

Copyright © 2019 The Authors. Computer Graphics Forum published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd. Reprinted under the Creative Commons CC-BY license with permission from Alexander Kumpf and Rüdiger Westermann.

C. Learning Generic Local Shape Properties for Adaptive Super-Sampling

Abstract of Paper We propose a novel encoder/decoder-based neural network architecture that learns view-dependent shape and appearance of geometry represented by voxel representations. Since the network is trained on local geometry patches, it generalizes to arbitrary models. A geometry model is first encoded into a sparse voxel octree of features learned by a network, and this model representation can then be decoded by another network in-turn for the intended task. We utilize the network for adaptive super-sampling in ray-tracing, to predict super-sampling patterns when seeing coarse-scale geometry. We discuss and evaluate the proposed network design, and demonstrate that the decoder network is compact and can be integrated seamlessly into on-chip ray-tracing kernels. We compare the results to previous screen-space super-sampling strategies as well as non-network-based world-space approaches.

Author Contributions The overall idea of the paper was developed by the first author in discussions with Prof. Dr. Rüdiger Westermann. The first author is responsible for the implementation as well as all studies presented in this work. The paper was composed by the first author under consideration of feedback as well as minor revisions by Prof. Dr. Rüdiger Westermann.

Note The paper has been published in a short paper track.

Copyright © 2022 The Author(s). Eurographics Proceedings © 2022 The Eurographics Association. Reprinted under Creative Commons Attribution CC-BY 4.0 with permission from Rüdiger Westermann.

D. Inverting the Feature Visualization Process for Feedforward Neural Networks

Abstract of Paper This work sheds light on the invertibility of feature visualization in neural networks. Since the input that is generated by feature visualization using activation maximization does, in general, not yield the feature objective it was optimized for, we investigate optimizing for the feature objective that yields this input. Given the objective function used in activation maximization that measures how closely a given input resembles the feature objective, we exploit that the gradient of this function w.r.t. inputs is—up to a scaling factor—linear in the objective. This observation is used to find the optimal feature objective via computing a closed form solution that minimizes the gradient. By means of Inverse Feature Visualization, we intend to provide an alternative view on a networks sensitivity to certain inputs that considers feature objectives rather than activations.

Author Contributions The overall idea of the paper was developed by the first author in discussions with Prof. Dr. Rüdiger Westermann. The first author is responsible for the implementation as well as all studies presented in this work. The paper was composed by the first author under consideration of feedback as well as minor revisions by Prof. Dr. Rüdiger Westermann.

Note The paper is hosted on arXiv.org but has not been peer-reviewed.

Copyright © 2020 The Authors. Reprinted as provided by arxiv.org with permission from Rüdiger Westermann.

In this thesis, we presented several techniques for processing and exploring large-scale datasets. Regarding high-resolution grid data, the $\mathcal{O}(n \log n)$ memory bottleneck of summed-volume tables (SVTs) has been resolved. By proposing a versatile data structure that evolves around performing a series of split operations, memory consumption can be controlled in the range of $\mathcal{O}(n)$ to $\mathcal{O}(n \log n)$ bits, with lower memory consumption implying lower access speed [RW21a]. Compared to alternative approaches, we achieve significantly reduced memory consumption at similar decoding performance, or vice versa.

Further, means to process high-dimensional data spaces have been proposed. We have improved upon visualizations of high-dimensional data via dimensionality reduction (DR), by incorporating the inherent randomness of several non-linear DR techniques in a novel robustness plot that depicts the stability of neighborhood relations in an ensemble of 2D point embeddings [RKW20]. Before our work, visualizations ignored the inherent randomness of frequently used DR techniques such as t-SNE by presenting the result of a single DR run only. Also, it has been shown that the high-dimensional space of local geometry patches can be represented by a task-specific, low-dimensional latent space encoding [RW22]. It is used to implement an effective, adaptive super-sampling scheme that can incorporate information about 3D geometric details even when the geometry is sampled at a coarse scale only. In contrast, screen-space-based, adaptive super-sampling schemes can only access coarse-scale 2D projections of geometry and thus are not able to detect all cases for which super-sampling improves image quality significantly.

Finally, we have suggested an approach to compare high-dimensional feature spaces in deep learning. As of yet, neural features have been successfully compared and classified by manual inspection of thousands of feature visualizations. However, we aim at the automatic matching of features with similar feature visualization. In our work, we have made a first step towards this goal by proposing the problem of inverse feature visualization (IFV) and showing that a novel method called gradient-based

inverse feature visualization (Grad-IFV) can identify the neurons for which a feature visualization has been generated [RW22].

Future Work

Our work can be extended in multiple ways. First, the performance of most methods can be improved by utilizing the compute-capabilities of graphics processing units (GPUs). Regarding the proposed SVT data structure, one can engineer cache-aware and/or GPU-accelerated encoding and decoding schemes, so that a) decoding can further benefit from massive parallelism and b) encoding can be realized in timings similar to state-of-the-art summed-area table (SAT) encoding. We believe that efficient SIMD decoding schemes are comparably easy to implement. One only has to compute indices where specific partial sums are stored in memory, then read these sums from global memory, and finally combine them with simple arithmetic operations. Specifically, we hypothesize that decoding will be memory-bound. However, throttling memory access patterns cannot be avoided in the decoding scheme since they are defined by the access patterns of the application utilizing the SVT. Encoding algorithms, on the other hand, have to be designed carefully. In our work, we propose an encoding scheme that creates a regular SVT as an intermediate step [RW21a]. However, paging of data between VRAM and larger RAM may be required. This can be avoided by performing split operations recursively on the GPU. Then, however, numerous, un-coalesced accesses to global memory may be difficult to avoid. The computation timings of robustness plots [RKW20] are dominated by the computation of k -order Voronoi diagrams. This step can be sped up significantly by discretizing the Voronoi diagram over a high-resolution grid and propagating the closest neighbors of grid cells by SIMD operations on the GPU [SKW09]. Also, we suggest speeding up our adaptive super-sampling scheme by performing sparse voxel octree (SVO) traversal on the GPU and realizing the decoder network via a fast Tensor Core implementation [WHW21; Mül+22].

Second, proposed methods can be applied in different contexts. For instance, we can think of deploying our SVT data structure in the process of creating acceleration structures for sparse scenes and data. Further, we believe that recursive split operations guided by a heuristic can significantly improve either performance or memory requirements of other data structures as well. For instance, instead of determining local histogram aggregations according to a mipmap structure as proposed by Al-Thelaya et al. [AAS21], we propose to aggregate histograms according to our representation. In doing so, a costly footprint assembly step can be avoided. Regarding our robustness plots [RKW20], we envision further applications in the fields of crowd analysis or fluid simulation, to identify groups of entities that remain close over time. In particular, our approach of clustering and selecting representative 2D embeddings may provide useful to identify repeating configurations in crowd analysis and to determine timestamps of fluid simulations at which significant topological changes occur. Regarding latent

space encodings of local geometry patches, we see applications in level of detail (LOD)-accelerated ray tracing applications, e.g. by approximating scattering events in distant, heterogeneous participating media. IFV, on the other hand, may be used to identify correlations between the existence of specific features and network performance. Then, highly effective features can be distilled in future training runs to obtain more consistent results and to speed up convergence.

Third, the limitations of proposed methods concerning theory and practice have to be addressed. As of yet, neither the space of optimal SVT representations nor the proposed heuristic is well understood. Given a prescribed constraint of either data access speed or memory, it would be interesting to identify formulas for lower and upper bounds of the respective other quantity when assuming either an optimal or a heuristic SVT representation. Also, we ask ourselves if there are any theoretical guarantees that the representations returned by the proposed heuristic are located in a specific band around the memory-performance trade-off curve introduced in our paper [RW21a]. One can think of designing better heuristics, providing tractable solutions to the combinatorial optimization problem of finding optimal parameter trees, or proving that the problem is NP-hard. Concerning latent space encodings of local geometry patches, further research is required to decrease the memory overhead of precomputing patch codes. Float quantization, varying the patch code dimension, and eliminating redundant patch codes are approaches we consider feasible. In addition to that, significant improvements on Grad-IFV have to be made. To reduce the fragility of Grad-IFV w.r.t. high frequency gradient signals, the Jacobian $\nabla N(\mathbf{I}^*)$ can be averaged over perturbed inputs. Here, perturbations can be introduced by either applying transforms such as rotations, translations, or scaling [OMS17], or by applying a small amount of Gaussian noise [Smi+17]. Further, Grad-IFV needs to be revisited for robust networks. As discussed in Sec. 2.7, recently it has been observed that explainable artificial intelligence (XAI) techniques perform notably better if investigated networks have been trained robustly. We believe that the same observation can be made concerning Grad-IFV, so that feature matching via Grad-IFV becomes viable.

Bibliography

- [AAS21] Khaled Al-Thelaya, Marco Agus, and Jens Schneider. “The Mixture Graph-A Data Structure for Compressing, Rendering, and Querying Segmentation Histograms”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 645–655. DOI: 10.1109/TVCG.2020.3030451.
- [AB18] Amina Adadi and Mohammed Berrada. “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)”. In: *IEEE Access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052.
- [Ade+18] Julius Adebayo et al. “Sanity Checks for Saliency Maps”. In: *Advances in Neural Information Processing Systems* 31. Ed. by Samy Bengio et al. Curran Associates, Inc., 2018, pp. 9505–9515.
- [Alc+19] Michael A. Alcorn et al. “Strike (With) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.
- [Api+21] Andrea Apicella et al. “A survey on modern trainable activation functions”. In: *Neural Networks* 138 (2021), pp. 14–32. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2021.01.026.
- [Aup07] Michaël Aupetit. “Visualizing distortions and recovering topology in continuous projection techniques”. In: *Neurocomputing* 70.7 (2007). *Advances in Computational Intelligence and Learning*, pp. 1304–1330. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2006.11.018.
- [Aur87] Franz Aurenhammer. “Power Diagrams: Properties, Algorithms and Applications”. In: *SIAM Journal on Computing* 16.1 (1987), pp. 78–96. DOI: 10.1137/0216006.

- [Aza+21] Mir M. Azad et al. “Medical diagnosis using deep learning techniques: A research survey”. In: *Annals of the Romanian Society for Cell Biology* 25.6 (2021), pp. 5591–5600.
- [BD05] Michael Balzer and Oliver Deussen. “Voronoi treemaps”. In: *IEEE Symposium on Information Visualization*. Oct. 2005, pp. 49–56. DOI: 10.1109/INFVIS.2005.1532128.
- [BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. “The Quickhull Algorithm for Convex Hulls”. In: *ACM Transactions on Mathematical Software* 22.4 (Dec. 1996), pp. 469–483. ISSN: 0098-3500. DOI: 10.1145/235815.235821.
- [Beh+16] Michael Behrisch et al. “Matrix Reordering Methods for Table and Network Visualization”. In: *Computer Graphics Forum* 35.3 (2016), pp. 693–716. DOI: 10.1111/cgf.12935.
- [Bel08] Harm J. W. Belt. “Word Length Reduction for the Integral Image”. In: *15th IEEE International Conference on Image Processing*. 2008, pp. 805–808. DOI: 10.1109/ICIP.2008.4711877.
- [BH86] Josh Barnes and Piet Hut. “A hierarchical $O(N \log N)$ force-calculation algorithm”. In: *Nature* 324.6096 (1986), pp. 446–449. ISSN: 1476-4687. DOI: 10.1038/324446a0.
- [Bha+21] Sweta Bhattacharya et al. “Deep learning and medical image processing for coronavirus (COVID-19) pandemic: A survey”. In: *Sustainable Cities and Society* 65, 102589 (2021). ISSN: 2210-6707. DOI: 10.1016/j.scs.2020.102589.
- [BHM10] Berkin Bilgic, Berthold K. P. Horn, and Ichiro Masaki. “Efficient integral image computation on the GPU”. In: *IEEE Intelligent Vehicles Symposium*. 2010, pp. 528–533. DOI: 10.1109/IVS.2010.5548142.
- [BPA22] Arnaud Bletterer, Frédéric Payan, and Marc Antonini. “Graph-based Computation of Voronoi Diagrams on Large-scale Point-based Surfaces”. In: *Eurographics – Short Papers*. Ed. by Nuria Pelechano and David Vanderhaeghe. The Eurographics Association, 2022. ISBN: 978-3-03868-169-4. DOI: 10.2312/egs.20221030.
- [BSB10] Amit Bhatia, Wesley E. Snyder, and Griff Bilbro. “Stacked Integral Image”. In: *IEEE International Conference on Robotics and Automation*. 2010, pp. 1530–1535. DOI: 10.1109/ROBOT.2010.5509400.
- [BTV06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “SURF: Speeded Up Robust Features”. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.
- [Cam+20] Nick Cammarata et al. “Thread: Circuits”. In: *Distill* (2020). DOI: 10.23915/distill.00024.

-
- [CAP20] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. “Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.
- [Car+19] Shan Carter et al. “Activation Atlas”. In: *Distill* (2019). DOI: 10.23915/distill.00015.
- [CC08] Michael A. A. Cox and Trevor F. Cox. “Multidimensional Scaling”. In: *Handbook of Data Visualization*. Berlin, Heidelberg: Springer, 2008, pp. 315–347. ISBN: 978-3-540-33037-0. DOI: 10.1007/978-3-540-33037-0_14.
- [Che+16] Tianqi Chen et al. “Training Deep Nets with Sublinear Memory Cost”. In: *arXiv e-prints* (Apr. 2016). arXiv: 1604.06174 [cs.LG].
- [Che+18] Peng Chen et al. “Efficient Algorithms for the Summed Area Tables Primitive on GPUs”. In: *IEEE International Conference on Cluster Computing (CLUSTER)*. 2018, pp. 482–493. DOI: 10.1109/CLUSTER.2018.00064.
- [Che+21] Zhiqin Chen et al. “DECOR-GAN: 3D Shape Detailization by Conditional Refinement”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 15735–15744. DOI: 10.1109/CVPR46437.2021.01548.
- [Che02] Chun-Houh Chen. “Generalized Association Plots: Information Visualization via Iteratively Generated Correlation Matrices”. In: *Statistica Sinica* 12.1 (2002), pp. 7–29.
- [Cis+17] Moustapha Cisse et al. “Parseval Networks: Improving Robustness to Adversarial Examples”. In: *arXiv e-prints* (Apr. 28, 2017). arXiv: 1704.08847v2 [stat.ML].
- [CMS13] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. “Density-Based Clustering Based on Hierarchical Density Estimates”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei et al. Berlin, Heidelberg: Springer, 2013, pp. 160–172. ISBN: 978-3-642-37456-2.
- [Cro84] Franklin C. Crow. “Summed-Area Tables for Texture Mapping”. In: *11th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH ’84*. New York, NY, USA: Association for Computing Machinery, 1984, pp. 207–212. ISBN: 0897911385. DOI: 10.1145/800031.808600.
- [Dai+17] Angela Dai et al. “ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.
- [Den+09] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
-

- [DFG99] Qiang Du, Vance Faber, and Max Gunzburger. “Centroidal Voronoi Tessellations: Applications and Algorithms”. In: *SIAM Review* 41.4 (1999), pp. 637–676. DOI: 10.1137/S0036144599352836.
- [DNJ20] Thomas Davies, Derek Nowrouzezahrai, and Alec Jacobson. “On the Effectiveness of Weight-Encoded Neural Implicit 3D Shapes”. In: *arXiv e-prints* (Sept. 2020). arXiv: 2009.09808 [cs.GR].
- [DSA20] Dan Dolonius, Erik Sintorn, and Ulf Assarsson. “UV-free Texturing using Sparse Voxel DAGs”. In: *Computer Graphics Forum* 39.2 (2020), pp. 121–132. DOI: 10.1111/cgf.13917.
- [DSW09] Christian Dick, Jens Schneider, and Rüdiger Westermann. “Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering”. In: *Computer Graphics Forum* 28.1 (2009), pp. 67–83.
- [Ehs+15] Shoaib Ehsan et al. “Integral Images: Efficient Algorithms for Their Computation and Storage in Resource-Constrained Embedded Vision Systems”. In: *Sensors* 15.7 (2015), pp. 16804–16830. DOI: 10.3390/s150716804.
- [Emo+18] Yutaro Emoto et al. “An Optimal Parallel Algorithm for Computing the Summed Area Table on the GPU”. In: *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 763–772. DOI: 10.1109/IPDPSW.2018.00123.
- [Eng+19] Logan Engstrom et al. “Adversarial Robustness as a Prior for Learned Representations”. In: *arXiv e-prints* (June 3, 2019). arXiv: 1906.00945v2 [stat.ML].
- [Erh+09] Dumitru Erhan et al. “Visualizing higher-layer features of a deep network”. In: *University of Montreal* 1341.3 (2009), p. 1.
- [Esp+21] Mateus Espadoto et al. “Toward a Quantitative Survey of Dimension Reduction Techniques”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.3 (2021), pp. 2153–2173. DOI: 10.1109/TVCG.2019.2944182.
- [Est+96] Martin Ester et al. “A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.
- [EUT19] Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. “ScalarFlow: A Large-Scale Volumetric Data Set of Real-World Scalar Transport Flows for Computer Animation and Machine Learning”. In: *ACM Transactions on Graphics* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356545.
- [Ewi+00] Jon P. Ewins et al. “Implementing an anisotropic texture filter”. In: *Computers & Graphics* 24.2 (2000), pp. 253–267. ISSN: 0097-8493. DOI: 10.1016/S0097-8493(99)00159-4.

-
- [Fen94] Peter M. Fenwick. “A New Data Structure for Cumulative Frequency Tables”. In: *Software: Practice and Experience* 24.3 (1994), pp. 327–336. DOI: 10.1002/spe.4380240306.
- [Fil+08] Maurizio Filippone et al. “A survey of kernel and spectral methods for clustering”. In: *Pattern Recognition* 41.1 (2008), pp. 176–190. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2007.05.018.
- [FLM14] Gabriele Facciolo, Nicolas Limare, and Enric Meinhardt-Llopis. “Integral Images for Block Matching”. In: *Image Processing On Line* 4 (2014), pp. 344–369. DOI: 10.5201/ipol.2014.57.
- [FSW09] Roland Fraedrich, Jens Schneider, and Rüdiger Westermann. “Exploring the Millennium Run - Scalable Rendering of Large-Scale Cosmological Datasets”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1251–1258. DOI: 10.1109/TVCG.2009.142.
- [Fuk80] Kunihiko Fukushima. “A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biol. Cybern.* 36 (1980), pp. 193–202.
- [FV17] Ruth C. Fong and Andrea Vedaldi. “Interpretable Explanations of Black Boxes by Meaningful Perturbation”. In: *IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 3449–3457. DOI: 10.1109/ICCV.2017.371.
- [Gar+21] Stephan J. Garbin et al. “FastNeRF: High-Fidelity Neural Rendering at 200FPS”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 14346–14355.
- [GAZ19] Amirata Ghorbani, Abubakar Abid, and James Zou. “Interpretation of Neural Networks Is Fragile”. In: *AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 3681–3688. DOI: 10.1609/aaai.v33i01.33013681.
- [GGB06] Michael Grabner, Helmut Grabner, and Horst Bischof. “Fast Approximated SIFT”. In: *Computer Vision – ACCV 2006*. Ed. by P. J. Narayanan, Shree K. Nayar, and Heung-Yeung Shum. Berlin, Heidelberg: Springer, 2006, pp. 918–927. ISBN: 978-3-540-32433-1.
- [GM04] Enrico Gobbetti and Fabio Marton. “Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models”. In: *Computers & Graphics* 28.6 (2004), pp. 815–826. ISSN: 0097-8493. DOI: 10.1016/j.cag.2004.08.010.

- [GM19] David Ganter and Michael Manzke. “An Analysis of Region Clustered BVH Volume Rendering on GPU”. In: *Computer Graphics Forum* 38.8 (2019), pp. 13–21. DOI: 10.1111/cgf.13756.
- [Goh+21] Gabriel Goh et al. “Multimodal Neurons in Artificial Neural Networks”. In: *Distill* (2021). DOI: 10.23915/distill.00030.
- [Goh19] Gabriel Goh. “A Discussion of ‘Adversarial Examples Are Not Bugs, They Are Features’: Two Examples of Useful, Non-Robust Features”. In: *Distill* (2019). DOI: 10.23915/distill.00019.3.
- [Goo+14] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.
- [GSS14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *arXiv e-prints* (Dec. 20, 2014). arXiv: 1412.6572v3 [stat.ML].
- [GV16] Patrick J. F. Groenen and Michel van de Velden. “Multidimensional Scaling by Majorization: A Review”. In: *Journal of Statistical Software* 73.8 (2016), pp. 1–26. DOI: 10.18637/jss.v073.i08.
- [Has+20] Jon Hasselgren et al. “Neural Temporal Adaptive Sampling and Denoising”. In: *Computer Graphics Forum* 39.2 (2020), pp. 147–155. DOI: 10.1111/cgf.13919.
- [He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.
- [Hed+21] Peter Hedman et al. “Baking Neural Radiance Fields for Real-Time View Synthesis”. In: *arXiv e-prints* (Mar. 26, 2021). arXiv: 2103.14645v1 [cs.CV].
- [Hen+05] Justin Hensley et al. “Fast Summed-Area Table Generation and its Applications”. In: *Computer Graphics Forum* 24.3 (2005), pp. 547–555. DOI: 10.1111/j.1467-8659.2005.00880.x.
- [Her+18] Hans Hersbach et al. *ERA5 hourly data on pressure levels from 1979 to present*. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). Accessed on 14-05-2022. 2018. DOI: 10.24381/cds.bd0915c6.
- [Her+20] Hans Hersbach et al. “The ERA5 global reanalysis”. In: *Quarterly Journal of the Royal Meteorological Society* 146.730 (2020), pp. 1999–2049. DOI: 10.1002/qj.3803.
- [HGK09] Max Hermann, Alexander Groß, and Reinhard Klein. “Interactive exploration of large event datasets in high energy physics”. In: *Journal of WSCG* 17.1–3 (2009), pp. 41–48.

-
- [HHS06] Vlastimil Havran, Robert Herzog, and Hans-Peter Seidel. “On the Fast Construction of Spatial Hierarchies for Ray Tracing”. In: *IEEE Symposium on Interactive Ray Tracing*. 2006, pp. 71–80. DOI: 10.1109/RT.2006.280217.
- [Hoh+20] Fred Hohman et al. “Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 1096–1106. DOI: 10.1109/TVCG.2019.2934659.
- [Hop82] John J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *National Academy of Sciences of the United States of America* 79.6953413 (Apr. 1982), pp. 2554–2558. ISSN: 1091-6490. DOI: 10.1073/pnas.79.8.2554.
- [Hop96] Hugues Hoppe. “Progressive Meshes”. In: *23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 99–108. ISBN: 0897917464. DOI: 10.1145/237170.237216.
- [HPD08] Mohamed E. Hussein, Fatih Porikli, and Larry Davis. “Kernel Integral Images: A Framework for Fast Non-Uniform Filtering”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587641.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
- [Hua+17] Gao Huang et al. “Densely connected convolutional networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.
- [HZ93] Geoffrey E. Hinton and Richard Zemel. “Autoencoders, Minimum Description Length and Helmholtz Free Energy”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993.
- [IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456.
- [Jia+20] Chiyu Jiang et al. “Local Implicit Grid Representations for 3D Scenes”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6000–6009. DOI: 10.1109/CVPR42600.2020.00604.
- [KA02] Andrew V. Knyazev and Merico E. Argentati. “Principal angles between subspaces in an A-based scalar product: Algorithms and perturbation estimates”. In: *SIAM Journal on Scientific Computing* 23.6 (2002), pp. 2008–2040. DOI: 10.1137/S1064827500377332.
-

- [KA21] Utku Kose and Jafar Alzubi. *Deep learning for cancer diagnosis*. Springer, 2021.
- [Kah+18] Minsuk Kahng et al. “ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 88–97. DOI: 10.1109/TVCG.2017.2744718.
- [Kal+17] Simon Kallweit et al. “Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks”. In: *ACM Transactions of Graphics* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130880.
- [Kár+21] Artúr I. Károly et al. “Deep Learning in Robotics: Survey on Model Structures and Training Strategies”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2021), pp. 266–279. DOI: 10.1109/TSMC.2020.3018325.
- [KB14] Diederik P. Kingma and Jimmy Ba. *Adam: A method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG]. 2014. arXiv: 1412.6980 [cs.LG].
- [KGB18] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.
- [Kin+19] Pieter-Jan Kindermans et al. “The (Un)reliability of Saliency Methods”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek et al. Cham: Springer International Publishing, 2019, pp. 267–280. ISBN: 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6_14.
- [KKR18] Alexandr Kuznetsov, Nima Khademi Kalantari, and Ravi Ramamoorthi. “Deep Adaptive Sampling for Low Sample Count Rendering”. In: *Computer Graphics Forum* 37.4 (2018), pp. 35–44. DOI: 10.1111/cgf.13473.
- [Kri09] Alex Krizhevsky. *Learning multiple layers of features from Tiny Images*. Tech. rep. University of Toronto, 2009.
- [Kum+18] Alexander Kumpf et al. “Visualizing Confidence in Cluster-based Ensemble Weather Forecast Analyses”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 109–119. ISSN: 1077-2626. DOI: 10.1109/tvcg.2017.2745178.
- [KW13] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *arXiv e-prints* (Dec. 2013). arXiv: 1312.6114 [stat.ML].
- [KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional scaling*. Sage Publications, Beverly Hills, 1978.

-
- [LA11] Sylvain Lespinats and Michaël Aupetit. “CheckViz: Sanity Check and Topological Clues for Linear and Non-Linear Mappings”. In: *Computer Graphics Forum* 30.1 (2011), pp. 113–125. DOI: 10.1111/j.1467-8659.2010.01835.x.
- [Lau07] Andrew Lauritzen. “Summed-Area Variance Shadow Maps”. In: *GPU Gems*. Vol. 3. Addison-Wesley Professional, 2007. Chap. 8.
- [Lee82] Der-Tsai Lee. “On k-Nearest Neighbor Voronoi Diagrams in the Plane”. In: *IEEE Transactions on Computers* C-31.6 (June 1982), pp. 478–487. ISSN: 0018-9340. DOI: 10.1109/TC.1982.1676031.
- [Li+17] Jundong Li et al. “Feature Selection: A Data Perspective”. In: *ACM Computing Surveys* 50.6 (Dec. 2017). ISSN: 0360-0300. DOI: 10.1145/3136625.
- [Lin20] Yen-Chen Lin. *Awesome Neural Radiance Fields*. 2020. URL: <https://github.com/yenchenlin/awesome-NeRF> (visited on 05/19/2022).
- [Liu+18] Hsueh-Ti D. Liu et al. “Beyond Pixel Norm-Balls: Parametric Adversaries using an Analytically Differentiable Renderer”. In: *arXiv e-prints* (Aug. 8, 2018). arXiv: 1808.02651v2 [cs.LG].
- [Liu+20a] Hsueh-Ti D. Liu et al. “Neural Subdivision”. In: *ACM Transactions of Graphics* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392418.
- [Liu+20b] Lingjie Liu et al. “Neural Sparse Voxel Fields”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 15651–15663.
- [Llo82] Stuart P. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.
- [LM09] Jan de Leeuw and Patrick Mair. “Multidimensional Scaling Using Majorization: SMA-COF in R”. In: *Journal of Statistical Software* 31.3 (2009), pp. 1–30. DOI: 10.18637/jss.v031.i03.
- [Low04] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94.
- [LRU85] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. “Statistically Optimized Sampling for Distributed Ray Tracing”. In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 61–68. ISSN: 0097-8930. DOI: 10.1145/325165.325179.
- [Lu+21] Yuzhe Lu et al. “Compressive Neural Representations of Volumetric Scalar Fields”. In: *Computer Graphics Forum* 40.3 (2021), pp. 135–146. DOI: 10.1111/cgf.14295.

- [Lue+03] David Luebke et al. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.
- [Maa14] Laurens van der Maaten. “Accelerating t-SNE using tree-based algorithms”. In: *Machine Learning Research* 15.1 (2014), pp. 3221–3245.
- [Mad+17] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *arXiv e-prints* (June 19, 2017). arXiv: 1706.06083v4 [stat.ML].
- [MH08] Laurens van der Maaten and Goeffrey Hinton. “Visualizing High-Dimensional Data Using t-SNE”. English. In: *Journal of Machine Learning Research* 9.nov (2008), pp. 2579–2605. ISSN: 1532-4435.
- [Mil+20] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 405–421. ISBN: 978-3-030-58452-8.
- [Mis13] Pushkar Mishra. “A New Algorithm for Updating and Querying Sub-arrays of Multidimensional Arrays”. In: *arXiv e-prints* (Nov. 24, 2013). arXiv: 1311.6093v6 [cs.DS].
- [Mit87] Don P Mitchell. “Generating Antialiased Images at Low Sampling Densities”. In: *14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 65–72. ISBN: 0897912276. DOI: 10.1145/37401.37410.
- [Mor+21] Eduardo F Morales et al. “A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning”. In: *Intelligent Service Robotics* 14.5 (2021), pp. 773–805.
- [MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going deeper into neural networks*. 2015.
- [MP99] Vitaly Maiorov and Allan Pinkus. “Lower bounds for approximation by MLP neural networks”. In: *Neurocomputing* 25.1 (1999), pp. 81–91. ISSN: 0925-2312. DOI: 10.1016/S0925-2312(98)00111-8.
- [MS15] Daniel Maturana and Sebastian Scherer. “VoxNet: A 3D Convolutional Neural Network for real-time object recognition”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928. DOI: 10.1109/IROS.2015.7353481.
- [Mül+21] Thomas Müller et al. “Real-time Neural Radiance Caching for Path Tracing”. In: *ACM Transaction of Graphics* 40.4 (Aug. 2021), 36:1–36:16. DOI: 10.1145/3450626.3459812.

-
- [Mül+22] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *arXiv e-prints* (Jan. 2022). arXiv: 2201.05989 [cs.CV].
- [MV15] Aravindh Mahendran and Andrea Vedaldi. “Understanding deep image representations by inverting them”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 5188–5196. DOI: 10.1109/CVPR.2015.7299155.
- [MV16] Aravindh Mahendran and Andrea Vedaldi. “Salient Deconvolutional Networks”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 120–135. ISBN: 978-3-319-46466-4.
- [Ngu+16] Anh Nguyen et al. “Synthesizing the preferred inputs for neurons in neural networks via Deep Generator Networks”. In: *Advances in Neural Information Processing Systems 29*. Ed. by Daniel D. Lee et al. Curran Associates, Inc., 2016, pp. 3387–3395.
- [Nie+20] Michael Niemeyer et al. “Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.
- [NSB04] Thomas Nocke, Heidrun Schumann, and Uwe Böhm. “Methods for the visualization of clustered climate data”. In: *Computational Statistics* 19.1 (Feb. 2004), pp. 75–94. ISSN: 1613-9658. DOI: 10.1007/BF02915277.
- [NYC19] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Understanding neural networks via Feature Visualization: A survey”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Cham: Springer International Publishing, 2019, pp. 55–76. ISBN: 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6_4.
- [Oka+00] Atsuyuki Okabe et al. *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley & Sons, 2000. ISBN: 0471986356.
- [Ola+18] Chris Olah et al. “The Building Blocks of Interpretability”. In: *Distill* (2018). DOI: 10.23915/distill.00010.
- [Ola+20] Chris Olah et al. “Zoom In: An Introduction to Circuits”. In: *Distill* (2020). DOI: 10.23915/distill.00024.001.
- [Ols04] Jacob Olsen. *Realtime procedural terrain generation - Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games*. 2004.
- [OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization”. In: *Distill* (2017). DOI: 10.23915/distill.00007.
-

- [Pap+16] Nicolas Papernot et al. “Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks”. In: *IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 582–597. DOI: 10.1109/SP.2016.41.
- [Par+19] Jeong Joon Park et al. “DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.
- [Pea01] Karl Pearson. “On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.
- [Pha+12] Thien Phan et al. “Performance-Analysis-Based Acceleration of Image Quality assessment”. In: *IEEE Southwest Symposium on Image Analysis and Interpretation*. 2012, pp. 81–84. DOI: 10.1109/SSIAI.2012.6202458.
- [Por05] Fatih Porikli. “Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, pp. 829–836. DOI: 10.1109/CVPR.2005.188.
- [Pri57] Robert C. Prim. “Shortest connection networks and some generalizations”. In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401. DOI: 10.1002/j.1538-7305.1957.tb01515.x.
- [Rau+17] Paulo E. Rauber et al. “Visualizing the Hidden Activity of Artificial Neural Networks”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 101–110. DOI: 10.1109/TVCG.2016.2598838.
- [RD18] Andrew Ross and Finale Doshi-Velez. “Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients”. In: *AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018).
- [RFS03a] Jaume Rigau, Miquel Feixas, and Mateu Sbert. “Entropy-based Adaptive Sampling”. In: *Graphics Interface*. Vol. 2. 3. 2003, pp. 79–87.
- [RFS03b] Jaume Rigau, Miquel Feixas, and Mateu Sbert. “Refinement Criteria Based on f-Divergences”. In: *14th Eurographics Workshop on Rendering*. EGRW ’03. Leuven, Belgium: Eurographics Association, 2003, pp. 260–269. ISBN: 3905673037.
- [RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

-
- [RKW20] Christian Reinbold, Alexander Kumpf, and Rüdiger Westermann. “Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques”. In: *Computer Graphics Forum* 39.1 (2020), pp. 333–346. DOI: 10.1111/cgf.13806.
- [RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.
- [RT21] Jay Roberts and Theodoros Tsiligkaridis. “Controllably Sparse Perturbations of Robust Classifiers for Explaining Predictions and Probing Learned Concepts”. In: *Machine Learning Methods in Visualisation for Big Data*. Ed. by Daniel Archambault, Ian Nabney, and Jaakko Peltonen. The Eurographics Association, 2021. ISBN: 978-3-03868-146-5. DOI: 10.2312/mlvis.20211072.
- [RW20] Christian Reinbold and Rüdiger Westermann. “Inverting the Feature Visualization Process for Feedforward Neural Networks”. In: *arXiv e-prints* (July 2020). arXiv: 2007.10757 [cs.LG].
- [RW21a] Christian Reinbold and Rüdiger Westermann. “Parameterized Splitting of Summed Volume Tables”. In: *Computer Graphics Forum* 40.3 (2021), pp. 123–133. DOI: 10.1111/cgf.14294.
- [RW21b] Christian Reinbold and Rüdiger Westermann. *Presentation of Parameterized Splitting of Summed Volume Tables*. 2021. URL: <https://youtu.be/JSHjLvIu1Y0?t=1392> (visited on 05/19/2022).
- [RW22] Christian Reinbold and Rüdiger Westermann. “Learning Generic Local Shape Properties for Adaptive Super-Sampling”. In: *Eurographics – Short Papers*. Ed. by Nuria Pelechano and David Vanderhaeghe. The Eurographics Association, 2022. ISBN: 978-3-03868-169-4. DOI: 10.2312/egs.20221032.
- [Sae+18] Nasir Saeed et al. “A Survey on Multidimensional Scaling”. In: *ACM Computing Surveys* 51.3 (May 2018). ISSN: 0360-0300. DOI: 10.1145/3178155.
- [Sel+17] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization”. In: *IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.



- [SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. In: *34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 3145–3153.
- [SH75] Michael Ian Shamos and Dan Hoey. “Closest-point problems”. In: *16th Annual Symposium on Foundations of Computer Science*. 1975, pp. 151–162. DOI: 10.1109/SFCS.1975.8.
- [Sit+18] Vincent Sitzmann et al. “DeepVoxels: Learning Persistent 3D Feature Embeddings”. In: *arXiv e-prints* (Dec. 3, 2018). arXiv: 1812.01024v2 [cs.CV].
- [Sit20] Vincent Sitzmann. *Awesome Implicit Neural Representations*. 2020. URL: <https://github.com/vsitzmann/awesome-implicit-representations> (visited on 05/19/2022).
- [SKB08] Faisal Shafait, Daniel Keysers, and Thomas M. Breuel. “Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images”. In: *Document Recognition and Retrieval XV*. Ed. by Berrin A. Yanikoglu and Kathrin Berkner. Vol. 6815. International Society for Optics and Photonics. SPIE, 2008, pp. 317–322. DOI: 10.1117/12.767755.
- [SKW09] Jens Schneider, Martin Kraus, and Rüdiger Westermann. “GPU-Based Real-Time Discrete Euclidean Distance Transforms With Precise Error Bounds”. In: *International Conference on Computer Vision Theory and Applications (VISAPP)*. 2009, pp. 435–442.
- [SLL20] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. “Visualizing the Impact of Feature Attribution Baselines”. In: *Distill* (2020). DOI: 10.23915/distill.00022.
- [Smi+17] Daniel Smilkov et al. “SmoothGrad: removing noise by adding noise”. In: *arXiv e-prints* (June 2017). arXiv: 1706.03825 [cs.LG].
- [Spr+14] Jost Tobias Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *arXiv e-prints* (Dec. 2014). arXiv: 1412.6806 [cs.LG].
- [SR17] Jens Schneider and Peter Rautek. “A Versatile and Efficient GPU Data Structure for Spatial Indexing”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 911–920. DOI: 10.1109/TVCG.2016.2599043.
- [Sri+14] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [Ste+56] Hugo Steinhaus et al. “Sur la division des corps matériels en parties”. In: *Bull. Acad. Polon. Sci* 1.804 (1956), p. 801.

-
- [STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *34th International Conference on Machine Learning*. Vol. 70. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3319–3328.
- [SVB18] Mehdi S. M. Sajjadi, Raviteja Vemulapalli, and Matthew Brown. “Frame-Recurrent Video Super-Resolution”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 6626–6634. DOI: 10.1109/CVPR.2018.00693.
- [SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. 2013. arXiv: 1312.6034 [cs.CV].
- [SW06] Jens Schneider and Rüdiger Westermann. “GPU-Friendly High-Quality Terrain Rendering”. In: *Journal of WSCG* 14.1-3 (2006), pp. 49–56. ISSN: 1213-6972.
- [Sze+13] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv e-prints* (2013). arXiv: 1312.6199 [cs.CV].
- [Sze+15] Christian Szegedy et al. “Going deeper with convolutions”. In: *IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
- [SZW19] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations”. In: *Advances in Neural Information Processing Systems*. Ed. by Hanna Wallach et al. Vol. 32. Curran Associates, Inc., 2019.
- [Tak+21] Towaki Takikawa et al. “Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11353–11362. DOI: 10.1109/CVPR46437.2021.01120.
- [Tap11] Ernesto Tapia. “A note on the computation of high-dimensional integral images”. In: *Pattern Recognition Letters* 32.2 (2011), pp. 197–201. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2010.10.007.
- [TdL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. “A Global Geometric Framework for Nonlinear Dimensionality Reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.
- [Tew+20] Ayush Tewari et al. “State of the Art on Neural Rendering”. In: *Computer Graphics Forum* 39.2 (2020), pp. 701–727. DOI: 10.1111/cgf.14022.
- [Thu+20] Nils Thuerey et al. “Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows”. In: *AIAA Journal* 58.1 (2020), pp. 25–36. DOI: 10.2514/1.J058291.
-

- [Tsi+18] Dimitris Tsipras et al. “Robustness May Be at Odds with Accuracy”. In: *arXiv e-prints* (May 30, 2018). arXiv: 1805.12152v5 [stat.ML].
- [TZN19] Justus Thies, Michael Zollhöfer, and Matthias Nießner. “Deferred Neural Rendering: Image Synthesis Using Neural Textures”. In: *ACM Transactions of Graphics* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3323035.
- [UBD13] Martin Urschler, Alexander Bornik, and Michael Donoser. “Memory Efficient 3D Integral Volumes”. In: *IEEE International Conference on Computer Vision (ICCV) Workshops*. June 2013.
- [Umm+19] Benjamin Ummenhofer et al. “Lagrangian fluid simulation with continuous convolutions”. In: *International Conference on Learning Representations*. 2019.
- [Vas+17] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by Isabelle Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
- [Ven07] Jarkko Venna. “Dimensionality reduction for visual exploration of similarity structures”. Doctoral thesis. 2007. ISBN: 978-951-22-8752-9.
- [VJ04] Paul Viola and Michael J. Jones. “Robust Real-Time Face Detection”. In: *International Journal of Computer Vision* 57.2 (2004), pp. 137–154.
- [VMD08] Vincent Vidal, Xing Mei, and Philippe Decaudin. “Simple Empty-Space Removal for Interactive Volume Rendering”. In: *Journal of Graphics Tools* 13.2 (2008), pp. 21–36. DOI: 10.1080/2151237X.2008.10129258.
- [Vor08] Georges Voronoi. “Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les paralléloèdres primitifs.” In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1908.134 (1908), pp. 198–287. DOI: 10.1515/crll.1908.134.198.
- [Wei+20] Sebastian Weiss et al. “Learning Adaptive Sampling and Reconstruction for Volume Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* (2020). DOI: 10.1109/TVCG.2020.3039340.
- [WHW21] Sebastian Weiss, Philipp Hermüller, and Rüdiger Westermann. “Fast Neural Representations for Direct Volume Rendering”. In: *arXiv e-prints* (Dec. 2, 2021). arXiv: 2112.01579v1 [cs.GR].
- [Wil83] Lance Williams. “Pyramidal Parametrics”. In: *10th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '83*. Detroit, Michigan, USA: Association for Computing Machinery, 1983, pp. 1–11. ISBN: 0897911091. DOI: 10.1145/800059.801126.

-
- [WK17] Eric Wong and Zico Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *arXiv e-prints* (Nov. 2, 2017). arXiv: 1711.00851v3 [cs.LG].
- [Won+18] Eric Wong et al. “Scaling provable adversarial defenses”. In: *Advances in Neural Information Processing Systems 31*. Ed. by Samy Bengio et al. Curran Associates, Inc., 2018, pp. 8400–8409. eprint: 1805.12514v2.
- [Xu+07] Qing Xu et al. “A Novel Adaptive Sampling by Tsallis Entropy”. In: *Computer Graphics, Imaging and Visualisation (CGIV 2007)*. 2007, pp. 5–10. DOI: 10.1109/CGIV.2007.10.
- [Yar+20] Lior Yariv et al. “Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance”. In: *Advances in Neural Information Processing Systems 33*. Ed. by Hugo Larochelle et al. Curran Associates, Inc., 2020, pp. 2492–2502.
- [Yia93] Peter N. Yianilos. “Data Structures and Algorithms for Nearest Neighbor”. In: *fourth annual ACM-SIAM Symposium on Discrete algorithms*. Vol. 66. SIAM. 1993, p. 311.
- [Yos+15] Jason Yosinski et al. “Understanding Neural Networks Through Deep Visualization”. In: *arXiv e-prints* (June 2015). arXiv: 1506.06579 [cs.CV].
- [ZF14] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1.
- [Zol+18] Michael Zollhöfer et al. “State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications”. In: *Computer Graphics Forum 37.2* (2018), pp. 523–550. DOI: 10.1111/cgf.13382.
- [ZSL18] Stefan Zellmann, Jürgen P. Schulze, and Ulrich Lang. “Rapid kd Tree Construction for Sparse Volume Data”. In: *Symposium on Parallel Graphics and Visualization. EGPGV’18*. Brno, Czech Republic: Eurographics Association, 2018, pp. 69–77.
- [Zwi+15] Matthias Zwicker et al. “Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering”. In: *Computer Graphics Forum 34.2* (2015), pp. 667–681. DOI: 10.1111/cgf.12592.

Parameterized Splitting of Summed Volume Tables

Christian Reinbold  and Rüdiger Westermann 

Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany

Abstract

Summed Volume Tables (SVTs) allow one to compute integrals over the data values in any cubical area of a three-dimensional orthogonal grid in constant time, and they are especially interesting for building spatial search structures for sparse volumes. However, SVTs become extremely memory consuming due to the large values they need to store; for a dataset of n values an SVT requires $\mathcal{O}(n \log n)$ bits. The 3D Fenwick tree allows recovering the integral values in $\mathcal{O}(\log^3 n)$ time, at a memory consumption of $\mathcal{O}(n)$ bits. We propose an algorithm that generates SVT representations that can flexibly trade speed for memory: From similar characteristics as SVTs, over equal memory consumption as 3D Fenwick trees at significantly lower computational complexity, to even further reduced memory consumption at the cost of raising computational complexity. For a $641 \times 9601 \times 9601$ binary dataset, the algorithm can generate an SVT representation that requires 27.0GB and $46 \cdot 8$ data fetch operations to retrieve an integral value, compared to 27.5GB and $1521 \cdot 8$ fetches by 3D Fenwick trees, a decrease in fetches of 97%. A full SVT requires 247.6GB and 8 fetches per integral value. We present a novel hierarchical approach to compute and store intermediate prefix sums of SVTs, so that any prescribed memory consumption between $\mathcal{O}(n)$ bits and $\mathcal{O}(n \log n)$ bits is achieved. We evaluate the performance of the proposed algorithm in a number of examples considering large volume data, and we perform comparisons to existing alternatives.

CCS Concepts

• **Information systems** → **Data structures**; • **Human-centered computing** → **Scientific visualization**;

1. Introduction

Summed Area Tables (SATs) are a versatile data structure which has initially been introduced to enable high-quality mipmapping [Cro84]. SATs store the integrals over the data values in quadratic areas of a two-dimensional orthogonal grid that start at the grid's origin. The entries in a SAT can be considered prefix sums, as they are computed via column- and row-wise one-dimensional prefix sums. With four values from a SAT the integral over any quadratic region can be obtained in constant time. The three-dimensional (3D) variant of SATs is termed Summed Volume Tables (SVTs). They are of special interest in visualization, since they can be used to efficiently build adaptive spatial search structures for sparse volumes. In particular, construction methods for kD-trees and Bounding Volume Hierarchies (BVHs) [VMD08, HHS06] can exploit SVTs to efficiently find the planes in 3D space where the space should be subdivided.

Fig. 1 shows a temperature snapshot in Rayleigh-Bénard convection flow of size $641 \times 9691 \times 9601$. To efficiently render this dataset via direct volume rendering algorithms, some form of adaptive spatial subdivision needs to be used to effectively skip empty space. However, the SVT from which such an acceleration structure can be computed requires 247.6GB of memory, so that only on computers with large memory resources all data can be stored

in main memory. While the input field is only of size $\mathcal{O}(n)$, the memory consumption of a SVT is of $\mathcal{O}(n \log n)$.

Alternative SVT representations such as 3D Fenwick trees [Fen94, Mis13, SR17] offer a memory-efficient intermediate data structure from which an adaptive space partition can be constructed. 3D Fenwick trees have a memory consumption of $\mathcal{O}(n)$ bits, yet recovering the integral values requires a number of $\mathcal{O}(\log^3 n)$ data fetch operations. For the example given in Fig. 1, a 3D Fenwick tree requires only 27.5GB of memory, but to obtain an integral value for a given volume $1512 \cdot 8$ fetches need to be performed.

For labelled datasets, SVTs can be used as an alternative to hierarchical label representations like the mixture-graph [ATAS21], to efficiently determine the number of labels contained in a selected sub-volume. Furthermore, SVTs can effectively support a statistical analysis of the data values in arbitrary spatial and temporal sub-domains. As another application of SVTs, we briefly sketch meteorological data analysis in Sec. 7. This includes the time- or memory-efficient computation of moving averages over selected sub-regions and time intervals.

1.1. Contribution

We propose an algorithm to generate SVT representations that can flexibly trade speed for memory. These representations build upon

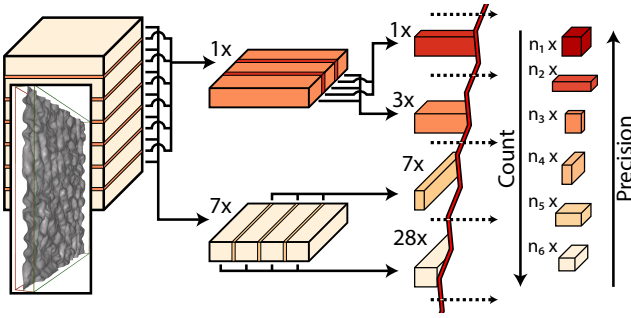


Figure 1: Schematic operation principle of splitting SVTs. A 3D array is hierarchically split into multiple high to low precision arrays (decreasing color saturation indicates decreasing precision), to obtain a data structure from which sums over axis-aligned subarrays can be computed. For the $641 \times 9601 \times 9601$ input volume obtained by a supercomputer simulation of a Rayleigh–Bénard convection, the SVT requires 247.6GB. Our algorithm generates SVT representations at 27.0GB or 71.2GB, requiring respectively 46 or 8 data fetch operations per prefix sum.

a recursive uni-axial partitioning of the domain and corresponding partial prefix sums, in combination with a hierarchical representation that progressively encodes this information. Since partial prefix sums require less bits to encode their values, the overall memory consumption can be controlled by the number and position of the performed domain splitting operations. By using different partitioning strategies, any prescribed memory consumption between $\mathcal{O}(n)$ bits and $\mathcal{O}(n \log n)$ bits can be achieved.

In principle, the algorithm proceeds in two phases: Firstly, for every possible SVT representation of a given volume an abstract *parameter tree* is constructed. This tree encodes the uni-axial split operations in a hierarchical manner, and it allows estimating both the memory consumption of the resulting SVT representation and the required data fetch operations for computing an integral value. Secondly, the tree is translated into the concrete SVT representation, by traversing the tree and performing the encoded operations.

To find a SVT representation according to a prescribed memory consumption or number of fetches, we propose a heuristic that generates a parameter tree which adheres to a given resource budget. This heuristic provides a close-to-optimal parameter tree for arbitrary budgets, over the entire spectrum ranging from memory-efficient yet compute-intense SVT representations to standard memory-exhaustive SVT representations with constant reconstruction time.

Our algorithm generates SVT representations with equal memory consumption as 3D Fenwick trees at significantly lower computational complexity. For the example in Fig. 1, we can construct a SVT representation that requires 27.0GB but requires only as few as $46 \cdot 8$ data fetch operations to retrieve an integral value, a decrease in data fetch operations of 97% compared to Fenwick trees. Our specific contributions are

- an abstract parameter tree representation that translates directly into a SVT representation,

- a capacity estimator for the memory and compute requirements of a given parameter tree,
- a heuristic that automatically provides a parameter tree that matches a prescribed capacity.

We analyze our proposed approach with respect to memory consumption and data fetch operations, and compare the results to those of alternative SVT representations. By using differently sized datasets, we demonstrate lower capacity requirements and improved scalability of our approach compared to others.

The paper is structured as follows: We first discuss approaches related to ours in the light of memory consumption and computational issues. After a brief introduction to the concept of SATs, we introduce the versatile data structure our approach builds upon. We demonstrate in particular the parameterization of this data structure to enable trading memory consumption for computational access efficiency. In Sec. 6, we then describe how to realize a concrete SVT representation that adheres to a user defined performance or memory budget. We evaluate our design choices and compare the obtained representations to alternative approaches in Sec. 7. We conclude our work with ideas for future work.

2. Related Work

SATs have been introduced by Crow [Cro84], as a data structure to quickly obtain integral values over arbitrary rectangular regions in 2D data arrays. Since then, SATs have found use in many computer vision and signal processing tasks such as object detection [BTVG06, VJ04, GGB06, Por05], block matching [FLML14], optical character recognition [SKB08] and region filtering [HPD08, Hec86, BSB10].

In computer graphics, SVTs are used to realize gloss effects [HSC*05], and in particular to accelerate the creation of spatial search structures for sparse scene or data representations. Havran et al. [HHS06] build a BVH / SKD-Tree hybrid acceleration structure for mesh data by discretizing the 3D domain and finding kd-splits in expected $\mathcal{O}(\log \log n)$. A SVT over the discretized domain is then used to evaluate the split cost function in constant time. Similarly, Vidal et al. [VMD08] propose to use SVTs to speed up cost function evaluations in a BVH construction process for voxelized volume datasets. In their work, the cost function requires the computation of bounding volumes over binary occupancy data. By running binary search on a SVT, this task can be solved in $\mathcal{O}(\log n)$ instead of $\mathcal{O}(n^3)$, where n is the side length of the volume. Ganter & Manzke [GM19] propose to use SVTs to cluster bounding volumes of small size before assembling them bottom-up into a BVH for Direct Volume Rendering (DVR). SVTs also allow to compute statistical quantities for arbitrarily large axis-aligned regions in constant time [PSCL12]. Thus, they facilitate interactive exploratory tasks in large scale volume datasets. The major drawback of SVTs is their memory consumption. Since prefix sums may span up to n elements, where n is the number of entries in a d -dimensional array, SVT entries require up to $\mathcal{O}(\log n)$ bits precision. This yields a total memory consumption of $\mathcal{O}(n \log n)$, where the original array is only of size $\mathcal{O}(n)$.

In Computer Graphics, classical Mip Mapping [Wil83] (or Rip Mapping in the anisotropic case) has been used for decades to ap-

proximately compute partial means (or equivalently sums) of textures in constant time and $\mathcal{O}(n)$ memory. Partial means of boxes with power of two side length are precomputed and then interpolated to approximate boxes with arbitrary side length. Belt [Bel08] proposes to apply rounding by value truncation to the input array before computing its corresponding SVT. By reducing the precision of the input, the SVT requires less bits of precision as well. To compensate for rounding error accumulation, the rounding routine is improved by considering introduced rounding errors of neighboring SVT entries. Clearly, this scheme cannot be used to reduce memory requirements for arrays of binary data. Although approximate schemes may suffice for imaging or computer vision tasks where small errors usually are compensated for, they are inappropriate in situations where hard guarantees are required (e.g. the support of a BVH has to cover all non-empty regions).

Exact techniques such as computation through the overflow [Bel08] or the blocking approach by Zellmann et al. [ZSL18] enforce a maximal precision bound per SVT entry and, thus, avoid the logarithmic increase in memory. The former approach simply drops all except for ℓ least significant bits and hence stores prefix sums modulo 2^ℓ . As long as queried boxes are small enough such that partial sums greater or equal than 2^ℓ are omitted, this method is exact. This approach excels in filtering tasks with kernels of prescribed box size. However, it is not applicable in situations where box sizes are either large or not known in advance.

Zellmann et al., on the other hand, brick the input array into 32^3 bricks and compute a SVT for each brick separately, hence they dub their method *Partial SVT*. As each prefix sum cannot sum over more than 32^3 elements, the amount of additional bits per entry is limited to 15 bits. However, when computing sums along boxes which do not fit into a single 32^3 brick, one value has to be fetched from memory for each brick intersecting the queried box. Since there still are $\mathcal{O}(n/(32^3)) = \mathcal{O}(n)$ many bricks, this approach scales equally poorly as summing up all values by iterating over the input array directly. More generally, any attempt to store intermediate sums with a fixed precision PREC scales poorly. In order to recover the "largest" possible sum of all array elements, one would have to add up at least $\mathcal{O}(n)/2^{\text{PREC}}$ factors of maximal size 2^{PREC} . To circumvent this issue, the authors propose to build a hierarchical representation of partial SVTs where the largest entries of each partial SVT form a new array that again is bricked and summed up partially. However, the authors admit that all bricks that overlap only partially with the queried box have to be processed at their current hierarchy level. Hence, the number of touched bricks reduces to the size of the box boundary. In the best case of cubes as boxes, the complexity still is $\mathcal{O}(n^{2/3})$ —which is impractical for $n \geq 1024^3$.

Ehsan et al. [ECRMM15] propose a technique that allows to compute arbitrary prefix sums by fetching a constant number of four values only through replacing each third row and column of a 2D array by their corresponding high precision SAT entries. By either adding an array entry to preceding SAT entries, or subtracting it from subsequent ones, the full SAT can be recovered. Their approach directly generalizes to 3D, requiring eight values instead. As a consequence, they only have to store 19 out of 27 SVT entries in high $\mathcal{O}(\log n)$ precision, reducing memory consumption by up

to 30%. However, total memory consumption still is in $\mathcal{O}(n \log n)$ and does not scale well.

3D Fenwick Trees as introduced by Mishra [Mis13] have beneficial properties with regard to both memory consumption and access. As shown by Schneider & Rautek [SR17], they require $\mathcal{O}(n)$ memory while allowing to compute prefix sums by summing up $\mathcal{O}(\log^3 n)$ values. In the 1D case [Fen94] this is achieved by recursively processing subsequent pairs of numbers such that the first number is stored verbatim; and the second one is summed up with the first one and then passed to the next recursion level. In the 3D case, this process generalizes to processing 2^3 -shaped blocks where one corner is stored verbatim and the other corners are processed recursively. Note that a complexity of $\mathcal{O}(\log^3 n)$ still yields numbers in the thousands for $n \geq 1024^3$ and above. Our approach improves significantly in this regard.

Memory Efficient Integral Volume (MEIV) by Urschler et al. [UBD13] first computes the full SVT and then partitions it into bricks of small size (brick sizes of 3^3 up to 12^3 were investigated). For each brick, MEIV stores its smallest prefix sum bo together with a parameter μ describing a one-parameter model for the brick entries subtracted by bo . The value of μ is determined by an optimization step performing binary search. Since the model cannot fit all block entries perfectly, the remaining error per entry is stored in a dynamic word length storage with smallest as possible precision. As a result, MEIV is able to decrease memory consumption exceptionally well with regard to the minimal overhead of fetching only two values from memory, namely some brick information and a value from the dynamic word length storage. Its clear downside is the increased construction time by fitting μ . In the authors' experiments, constructing the MEIV representation with optimal block size for the *largeRandomVolume* dataset takes 75 times longer than for the regular SVT. Further, MEIV cannot give any memory guarantees as the final memory consumption is sensitive to the chosen brick size as well as the actual dataset. Compared to MEIV, we obtain the same savings in memory by allowing 6 instead of 2 fetches from memory, and—more importantly—memory requirements of our approach are known before actual encoding. Further, our approach is able to flexibly adapt memory requirements in the full range of $\mathcal{O}(n)$ to $\mathcal{O}(n \log n)$ respecting the user's need, and thus still can be used whereas other approaches (especially Ehsan and MEIV) run out of memory.

3. Summed Volume Tables

We now briefly describe the basic concept underlying SVTs. For the sake of clarity, we do so on the example of a SAT, the 2D counterpart of a SVT, before we extend the concept to an arbitrary number of dimensions. Note that we use **1-based indexing** whenever accessing arrays during the course of the paper.

Given a two-dimensional array F of scalar values, an entry (x, y) of its corresponding SAT is computed by summing up all values contained in the rectangular subarray that is spanned by the indices $(1, 1)$ and (x, y) , that is

$$\text{SAT}_F[x, y] := \sum_{x' \leq x, y' \leq y} F[x', y'].$$

If the values of a SAT are precomputed, partial sums of F for arbitrary rectangular subarrays can be computed in constant time, by making use of the inclusion-exclusion principle. Instead of reading and adding up values of F along the whole region spanned by $(x_1 + 1, y_1 + 1)$ and (x_2, y_2) , it suffices to read the SAT-values at the corners of the selected subarray. It holds that

$$\sum_{x_1 < x' \leq x_2, y_1 < y' \leq y_2} F[x', y'] = \text{SAT}_F[x_1, y_1] + \text{SAT}_F[x_2, y_2] - \text{SAT}_F[x_1, y_2] - \text{SAT}_F[x_2, y_1]$$

with $\text{SAT}_F(x, y)$ set to zero if $x = 0$ or $y = 0$. Thus, SATs reduce the summation of $(x_2 - x_1) \cdot (y_2 - y_1)$ values to a summation of four values. The concept of SATs extends to any number of dimensions. Given a d -dimensional array F , the corresponding d -dimensional SVT is realised by

$$\text{SVT}_F[v_1, v_2, \dots, v_d] := \sum_{v'_i \leq v_i} F[v'_1, v'_2, \dots, v'_d].$$

Partial sums of hyperboxes (line segments in 1D, rectangles in 2D, cuboids in 3D, and so on) can be computed by evaluating a SVT at the 2^d corner points of that hyperbox. In the special case of $d = 1$, a SVT stores prefix sums of a 1D array. The entries in a SVT can be interpreted as d -dimensional prefix sums regarding axis-aligned volumes.

4. Hierarchical SVT data structure

We now introduce a hierarchical approach that allows identifying the intermediate representation to store a SVT so that a prefix sum can be computed with as little as possible additional compute under a prescribed memory budget. Here we assume that the input array F stores non-negative integral numbers. Thus, negative entries need to be eliminated by appropriate shifting, and floating point values need to be rescaled. While shifting does not affect accuracy, since the sign bit can be reused, floating point numbers cannot be rescaled to integers in a reasonable way if their value range is too large. In this case, however, computing partial sums even with the plain SVT is likely to fail due to numerical errors introduced by adding and subtracting potentially large prefix sums.

We require the following notation: An array F is said to have shape $n \in \mathbb{N}^d$ if and only if it is d -dimensional of shape $n_1 \times n_2 \times \dots \times n_d$. We define the size of n by $|n| := \prod_{i=1}^d n_i$. In particular, an array of shape n has $|n|$ elements. Further, for any multi index $v \in \mathbb{N}^d$ and integers $k \in \{1, \dots, d\}$, $i \in \mathbb{N}$, we denote by $v|_{k=i}$ the multi index that is obtained by replacing the k -th component of v by i . When accessing array elements via a multi index, $F[v]$ is a shorthand notation for $F[v_1, \dots, v_d]$.

Our proposed intermediate representation (i.e. a data structure) evolves around the concept of splitting the input array F of shape $n \in \mathbb{N}^d$ into a small array F_a of precomputed, high precision aggregates and a set $\{F_{s_0}, F_{s_1}, \dots\}$ of low precision subarrays such that any prefix sum can be efficiently computed from one prefix sum of the aggregate array and one prefix sum of a single subarray. Fig. 2 illustrates an exemplary split of a 3D input array. The aggregates are obtained by summing values of F along bands following one of the d dimensions, let us say the k -th one, which is called the *split dimension*. Each band is dissected into multiple segments by

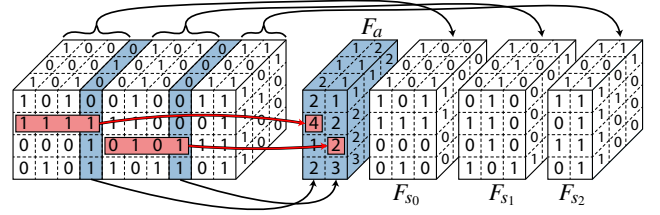


Figure 2: Splitting a $10 \times 4 \times 3$ block along the largest dimension into the aggregate array F_a and three subarrays $F_{s_0}, F_{s_1}, F_{s_2}$. Cut position are marked in blue. Red blocks and arrows indicate how two entries of F_a are formed by summation.

cutting at positions $c_1 < c_2 < \dots < c_\ell$, called *split positions*, along the split dimension. For each segment, its values of F are summed to form an aggregate. All aggregates are arranged in the aggregate array F_a of shape $n|_{k=\ell}$. More precisely, it is

$$F_a[v] = \sum_{i=c_{k-1}+1}^{c_k} F[v|_{k=i}],$$

where $v \in \mathbb{N}^d$ and $c_0 := 0$. The prefix sums of F_a correspond to prefix sums of F ending at corners $v \in \mathbb{N}^d$ with $v_k \in \{c_1, \dots, c_\ell\}$.

To enable the computation of any prefix sum of F , additional $\ell + 1$ subarrays $F_{s_i} \subseteq F$ of shape $n|_{i=c_{i+1}-c_i-1}$ with $F_{s_i}[v] = F[v|_{k=c_i+v_k}]$ are defined. Here, i ranges from 0 to ℓ with $c_{\ell+1} := n_k + 1$. Any prefix sum of F up to the corner $v \in \mathbb{N}^d$ now can be computed as follows: The index of the last split position not succeeding v , i.e., $i = \max(\{m \mid c_m \leq v_k\} \cup \{0\})$, and the subarray offset $j = v_k - c_i$ are determined. Then it is

$$\text{SVT}_F[v] = \text{SVT}_{F_a}[v|_{k=i}] + \text{SVT}_{F_{s_i}}[v|_{k=j}]. \quad (1)$$

Note that due to $\ell + \sum_{i=0}^{\ell} (c_{i+1} - c_i - 1) = c_{\ell+1} - c_0 - 1 = n_k$, the number of values stored in F equals the numbers of values stored in F_a and all subarrays. Since values in SVTs of subarrays arise as sums over only a fraction of values of F , they require less bits of precision than values in the SVT of F . Thus, storing SVTs after applying the splitting operation comes with memory savings at the cost of one additional memory fetch and addition per prefix sum query on F . The memory savings can be reinforced at the cost of more fetches by recursively applying the split process to the newly acquired arrays F_a and all F_{s_i} until a certain termination condition holds. Then, each terminal array of small shape is stored by encoding either its entries (verbatim), or the entries of its SVT in fixed precision. The result is a split hierarchy of which an example is shown in Fig. 3. To compute a prefix sum from this representation, Eq. (1) is applied recursively up to the point where a prefix sum can be derived from a terminal array stored in memory.

4.1. The parameter tree

We describe a specific split hierarchy by means of a *parameter tree*. When splitting the input array F , we encode split dimension as well as split positions into the root node of the parameter tree. If a newly acquired array (aggregate array or subarray) is split further, a

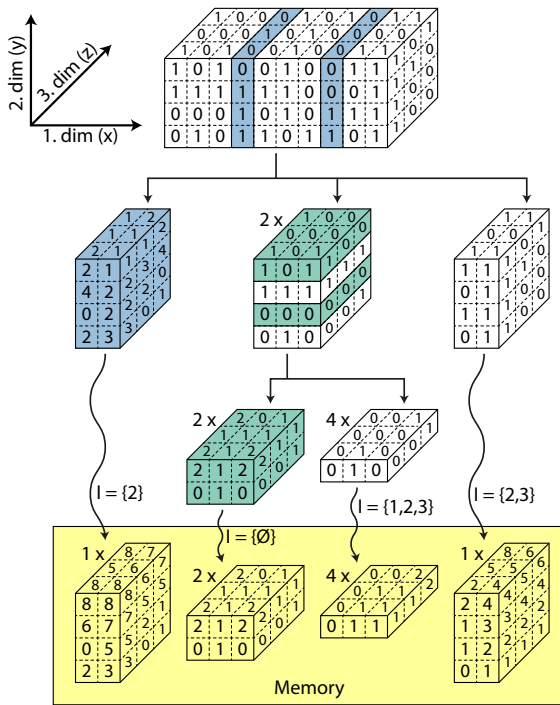


Figure 3: Sample split hierarchy of a $10 \times 4 \times 3$ binary input array. We show aggregate arrays and subarrays generated during the recursive split process. Whenever two or more subarrays of similar shape form in a split operation, a multiplier in the top left corner of a block indicates how many arrays of its shape arise. The block's filling with numbers matches the first (i.e. leftmost/undermost) of its associated subarrays. The other subarrays of similar shape may contain different numbers. As a final step indicated by wavy arrows, each terminal array is processed by computing cumulative sums according to the leaf parameter I (see Sec. 4.1). The result is stored in memory.

parameter subtree representing its subsequent split process is built and attached to the root node. If a newly acquired array is terminal, a leaf node describing its memory layout is created and attached instead.

In a naive implementation, the parameter quickly becomes unmanageable since it is branching with a factor that scales with the number of subarrays per split. By recursively splitting all subarrays of similar shape in the same way, one can collapse all of their corresponding subtrees to a single one and thus reduce the branching factor to the number of different subarray shapes occurring in a split, plus one for the aggregate array. Hence, we constrain the branching factor by requiring as few as possible different subarray shapes. This can be achieved by specifying a fixed subarray size z along the split dimension and placing split positions accordingly with equal spacing. However, we have to resolve alignment issues if $z + 1$ is not a divisor of the length n_k of the split dimension minus one. Overall, we experimented with three different alignment strategies:

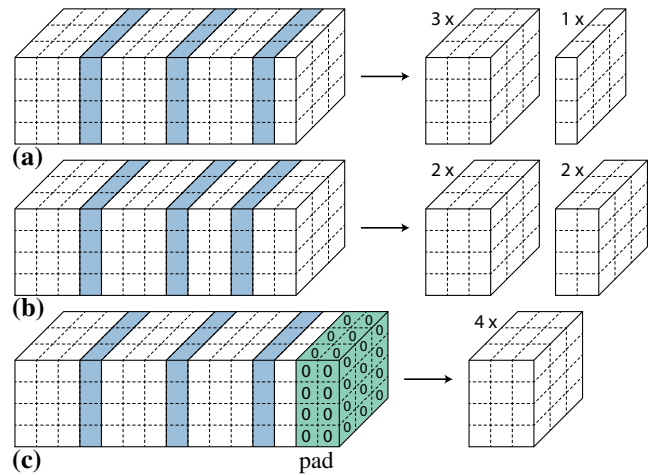


Figure 4: Splitting a $13 \times 4 \times 3$ array along the longest dimension into subarrays of size 3 with (a) *at_end*, (b) *distributed* and (c) *pad* alignment. Split positions are highlighted on the left. The number and shape of resulting subarrays is shown on the right.

at_end: The last subarray remains "incomplete" and thus has a size smaller than z along the split dimension.

distributed: A slice is removed from as many subarrays as one would have to pad in order to expand the last subarray to size z along the split dimension.

pad: The last subarray is padded with zeros until size z is reached.

Figure 4 depicts the results of all alignment strategies when splitting a $13 \times 3 \times 3$ field with fixed subarray size of 3 along the first dimension. During our experiments we noticed no difference in quality when generating SVT representations with either *at_end* or *distributed* aligned splits. Further, both yield subarrays of at most two different shapes, thus restricting the branching factor of the recursion to 3. *Pad* alignment incurs an additional memory overhead of up to 10%. In return it guarantees a unique subarray shape, reducing the branching factor to 2. This property may be favourable when engineering massively parallel en- & decoding schemes in future work. In the scope of this paper we decided to utilize *distributed* aligned splits.

In summary, a split now is defined by the split dimension k and subarray size z that allows to infer the split positions according to the *distributed* alignment strategy. Both values are encoded into an internal tree node representing the split. A leaf node, on the other hand, contains a set of dimension indices $I \subseteq \{1, \dots, d\}$ which describe along which dimensions array values are cumulated before finally storing each cumulated value in memory. The special cases of storing verbatim or SVT entries are represented by $I = \emptyset$ and $I = \{1, \dots, d\}$, respectively. Fig. 5 shows the parameter tree describing the split hierarchy of Fig. 3. Note that a tree node may represent more than one array by collapsing subtrees of similarly shaped subarrays. For instance, two arrays of shape $3 \times 4 \times 3$ are represented by the "k= 2, z= 1" internal node, and the memory layout of the four terminal arrays of shape $3 \times 1 \times 3$ is given by the "I = {1,2,3}" node.

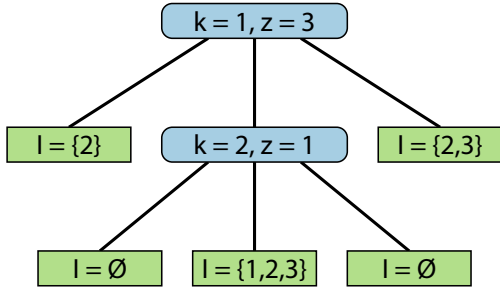


Figure 5: A parameter tree of depth 2 describing the split hierarchy of Fig. 3. It has two internal (blue) and five leaf nodes (green). The leftmost subtree of each internal node describes the split hierarchy of the aggregate array whereas the remaining subtrees describe the split hierarchy of subarrays for two different subarray shapes. Text in nodes indicate encoded parameters. Note that the lower right leaf node can be chosen arbitrarily since for the 3D shape given in Fig. 3 there is only one unique subarray shape arising from the lower split.

4.2. The conjugate trick

To reduce the numbers of split positions and thus the number of high precision aggregates by one half (without changing the subarray size), we are generalizing the technique described by Ehsan et al. [ECRMM15]. Instead of adding up a subarray prefix sum and the aggregate prefix sum at the preceding split position as in Eq. (1), one can obtain the same result by subtracting the prefix sum of a flipped subarray from the aggregate prefix sum at the subsequent split position. If F_{s_i} is the $(i+1)$ -th subarray with entries $F_{s_i}[v] = F[v]_{k=c_i+v_k}$, we denote with $F_{s_i}^*$ its *conjugate* that is obtained by shifting by one and mirroring along the split dimension, i.e. $F_{s_i}^*[v] = F[v]_{k=c_{i+1}+1-v_k}$. Then we have

$$\text{SVT}_F[v] = \text{SVT}_{F_a}[v]_{k=i} - \text{SVT}_{F_{s_i}^*}[v]_{k=j}, \quad (2)$$

where $i = \min\{m \mid c_m \geq v_k\}$ and $j = c_i - v_k$.

By replacing every second subarray by its conjugated version, one out of two split positions become superfluous. An exemplary split resulting from this process is shown in Fig. 6. Whenever a corner v is located in a subarray with odd index, Eq. (2) is used to compute the prefix sum, and Eq. (1) otherwise. If the last subarray of a split is a conjugate one, a split position at n_k (the last possible position) is added to ensure that a subsequent split position always exists. Allowing for both addition and subtraction and thus halving the size of F_a generally improves the final SVT representation by a more shallow split hierarchy and/or smaller block sizes at leaf levels. This is advantageous as smaller SVT leaves require less precision per entry, and smaller verbatim leaves require less additions to obtain a prefix sum.

5. Analysis of the parameter tree

If the shape of the input array F as well as its largest possible entry (usually of the form $2^{\#\text{bits}} - 1$) is known in advance, all relevant properties of a SVT representation of F can be derived via

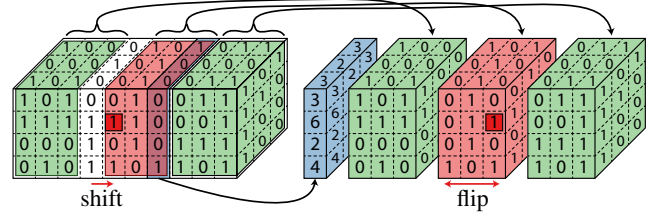


Figure 6: Splitting a $10 \times 4 \times 3$ array by permitting subtraction. Compared to Fig. 2, the first split position is introduced after two subarrays instead of one, and the second (red) subarray is conjugated by first shifting by one and then flipping. Note how the boxed 1 changes position. Subarrays in green are not conjugated.

a top-down-top traversal of the corresponding parameter tree describing the split hierarchy. When descending the tree, information about shapes and largest possible entries is propagated according to the split parameters k, z . Note that whereas the largest possible entry for subarrays can be taken over from the array being split, for aggregate arrays, an additional factor of $2z+1$ has to be applied. It matches the number of values that are summed up to compute a single entry of the aggregate array when utilizing the conjugate trick (see Sec. 4.2).

5.1. Memory requirements

Let T be the parameter tree. If T consists of a single, terminal node with dimension indices I , the largest possible entry that will be stored in memory is given by $m \cdot \prod_{i \in I} n_i$, where n is the array shape at the terminal node and m is the array's largest possible entry. Consequently, we have

$$\text{MEM}(T) = |n| \cdot \lceil \log_2(1 + m \cdot \prod_{i \in I} n_i) \rceil.$$

If the root node of T is an internal node, let T_a be the subtree describing the representation of the aggregate array F_a , and let T_{s_1} and T_{s_2} be the two subtrees describing representations for the two distinct subarray shapes. Then, the memory consumption can be computed as

$$\text{MEM}(T) = \text{MEM}(T_a) + \lambda_1 \cdot \text{MEM}(T_{s_1}) + \lambda_2 \cdot \text{MEM}(T_{s_2}),$$

where λ_i is the number of subarrays with shape represented by T_{s_i} .

The memory required for storing the parameter tree itself is negligible. Even for large datasets of GB-scale, the whole parameter tree is of KB-size. If the parameter tree is fixed beforehand and baked into the encoding & decoding algorithm, it does not need to be stored at all.

5.2. Estimation of fetch operations

The parameter tree T exposes an upper bound for the number of fetch operations required to compute a prefix sum. We call this bound the *fetch estimate* for T and denote it by $\text{FETCH}(T)$. If the root node of T is terminal with dimension indices I , we require

$$\text{FETCH}(T) = \prod_{i \in \{1, \dots, d\} \setminus I} n_i$$

fetches to compute a prefix sum, with n being the array shape at the terminal node. If the root node of T is internal, we have

$$\text{FETCH}(T) = \text{FETCH}(T_a) + \max(\text{FETCH}(T_{s_1}), \text{FETCH}(T_{s_2})) \quad (3)$$

with the same notation as in Sec. 5.1. Since computing a prefix sum according to Eq. (1, 2) is performed by querying a prefix sum of the aggregate array and one subarray, the fetch estimate is an upper bound for the number of fetch operations—however, not necessarily the minimal one. In the appendix, we present a method for computing a tighter bound that in our experiments is lower by 3% on average and 24% at most. All fetch counts presented in this paper are computed with the tighter bound instead of the fetch estimate.

A very coarse upper bound for the number of compute operations per prefix sum can be given by four times the number of fetches. Since memory access is of magnitudes slower than simple arithmetic operations, we conclude that computing prefix sums is memory-bound. Hence, we use the number of fetch operations as performance indicator.

5.3. Update costs

Whenever a value of F is modified, a single aggregate of F_a as well as one value of the subarray containing the modified value have to be updated. Hence, the cost $\text{UPDATE}(T)$ for updating the SVT representation can be described by the same recursion formula (3) as for the fetch estimate. At terminal nodes however, update costs are computed by

$$\text{UPDATE}(T) = \prod_{i \in I} n_i$$

since an entry of a terminal array of shape n can be part of up to $\prod_{i \in I} n_i$ sums stored in memory.

5.4. Construction costs

In the supplement, we show that each value stored in memory by our SVT representation is a certain partial sum of the input array F . Hence, all stored values can be efficiently determined by computing the classical SVT of F for instance via GPU computing, and then sampling a partial sum from the SVT for each stored value. Since our representation stores n values at leaf nodes, and SVTs can be computed in $\mathcal{O}(n)$ and sampled in $\mathcal{O}(1)$, the overall runtime-complexity of the construction algorithm is $\mathcal{O}(n)$. Due to the fundamental assumption that $\mathcal{O}(n \log n)$ of main memory is not available, we propose to realize the classical SVT via bricking strategies falling back to larger, but slower memory (e.g. persistent storage). After construction, partial sums can be queried by fetching data from the constructed SVT representation that is stored in (fast) main memory.

6. Identification of optimal parameter trees

Our split hierarchy design opens up a high-dimensional search space for SVT representations, with the parameters trees being elements in the space of parameters defining the hierarchy. Ideally,

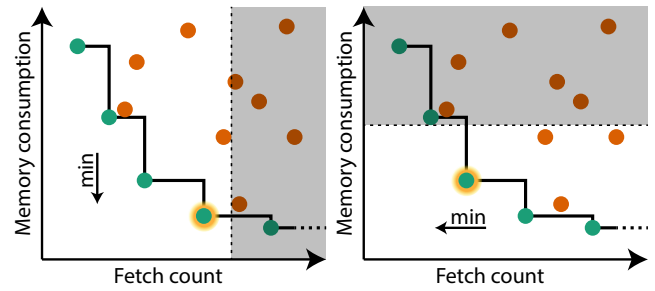


Figure 7: Schematic illustration of parameter tree search spaces. Each dot represents a parameter tree of certain fetch count and memory consumption. Optimal parameter trees are colored in green and define the memory-performance trade-off curve shown in black. By restricting the search space in one quantity and optimizing for the other, the optimum is uniquely determined (highlighted dot).

querying this search space for SVT representations yields a parameter tree instance that minimizes both the fetch count and the memory consumption with respect to input arrays of fixed shape and precision. However, representations with low fetch count have a high memory consumption and vice versa. To obtain a well defined optimization problem, we restrict the search space to representations that do not exceed a prescribed budget of *either* the number of fetch operations *or* the memory consumption, and then ask for the SVT representation that minimizes the respective other quantity. The resulting parameter trees follow a memory-performance trade-off curve as shown in Fig. 7.

The parameters which define a parameter tree are all discrete quantities, so that we are facing a combinatorial optimization problem. Even though we do not give a formal proof here, we believe that this problem is NP-hard. Thus, we propose a heuristic \mathcal{H} that receives the shape n of the input array as well as a control parameter λ and returns a parameter tree $\mathcal{H}(\lambda, n)$ that is close to the memory-performance trade-off curve. Increasing [decreasing] λ typically results in parameter trees with higher [lower] fetch count and lower [higher] memory consumption. In particular, this design allows finding a beneficial parameter tree for a prescribed budget B of either fetches or—more interestingly—memory. It is achieved by defining the function

$$f(\lambda) = \begin{cases} \text{FETCH}(\mathcal{H}(\lambda, n)) - B & \text{if fetch budget} \\ \text{MEM}(\mathcal{H}(\lambda, n)) - B & \text{if memory budget} \end{cases}$$

and applying a root-finding method such as the bisection method to find λ with $f(\lambda) \leq 0$ being close to zero. Then, $\mathcal{H}(\lambda, n)$ is a parameter tree that, on the one hand, minimizes the unconstrained quantity, and exhausts the given budget on the other hand.

In the algorithmic formulation of the heuristic, λ represents a threshold for the fetch estimate described in Sec. 5.2. It is guaranteed that the fetch estimate of $\mathcal{H}(\lambda, n)$ does not exceed λ . We achieve this by the following procedure: If λ equals one, the heuristic returns the parameter tree representing a classical SVT; that is a single leaf node with $I = \{1, \dots, d\}$. If, on the other hand, λ is at least as large as the shape product $|n|$, it returns a single leaf

node with setting $I = \emptyset$ —that is the verbatim representation. For $1 < \lambda < |n|$, the heuristic builds an internal node. The split dimension k is chosen such that $n_k = \max(n_1, \dots, n_d)$. The subarray size z is derived from λ according to an interpolation function that is manually defined to match the structure of optimal parameter trees for small array shapes. These were computed once by a Branch-and-Bound strategy that performs an exhaustive search.

The split parameters k and z define the shapes $n^{(a)}$, $n^{(s_1)}$, $n^{(s_2)}$ of the aggregate array and the subarrays. Thus, we can use the heuristic recursively to compute the subtrees of the internal node. To do so, we define control parameters λ_a , λ_s and set the aggregate subtree to $\mathcal{H}(\lambda_a, n^{(a)})$ and both subarray subtrees to $\mathcal{H}(\lambda_s, n^{(s_1)})$ and $\mathcal{H}(\lambda_s, n^{(s_2)})$ respectively. By requiring $\lambda_a, \lambda_s \geq 1$ and $\lambda_a + \lambda_s \leq \lambda$, we assure that the recursion terminates and that the fetch estimate of the final parameter tree does not exceed λ . This is due to Eq. (3) and the assumption that the heuristic already satisfies this guarantee for recursive calls. In order to find a reasonable choice for λ_a and λ_s , we again analyzed optimal parameter trees and noticed that the ratio of the fetch estimate $\text{FETCH}(T_a)$ of the aggregate subtree to the fetch estimate $\text{FETCH}(T)$ of the whole tree correlates to the ratio of the number of split positions $(n^{(a)})_k$ to the length n_k of the split dimension. While the latter ratio can be computed from the split parameters, the first is unknown yet. However, by assuming that the observed correlation applies for arbitrary array shapes, we can derive an estimate for the first ratio. Now, the exact value for λ_a is obtained by matching λ_a/λ to the estimate of the first ratio. Then, λ_s is computed by subtracting λ_a from λ . Pseudo code for the heuristic is given in the appendix.

7. Evaluation

The following evaluation sheds light on a) the quality of the heuristic to find an optimal parameter tree (Sec. 6) and b) on the properties of our derived SVT representations compared to alternative approaches, such as MEIV by Urschler et al. [UBD13], 3D Fenwick Trees by Mishra [Mis13], Partial SVTs by Zellmann et al. [ZSL18], and the approach of Ehsan et al. [ECRMM15].

All our experiments were run on a server architecture with 4x Intel Xeon Gold 6140 CPUs with 18 cores @ 2.30GHz each. Although we do not exploit any degree of parallelism, the generation of parameter trees with the proposed heuristic requires between 8.5s seconds for the smallest and 45.4s for the largest dataset (see Table 1). Timings reflect convergence speed of the bisection method used during parameter tree search as well as the cost for evaluating the heuristic in each bisection step. Parameter tree generation for single digit fetch counts is fast since the runtime of the heuristic correlates with the size of the parameter tree—and trees are very shallow in that scenario.

As the parameter tree has to be created only once for a fixed array shape and precision, we consider the runtime of the heuristic as negligible. Finding globally optimal parameter trees, however, is not tractable even for MB-scale datasets. The Branch-and-Bound strategy for precomputing optimal parameter trees already takes 12.5 hours for a 64^3 dataset, clearly necessitating the use of a heuristic.

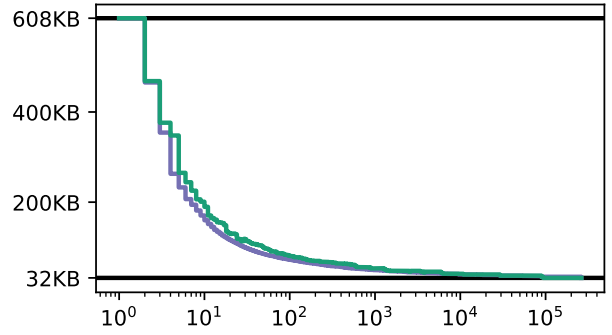


Figure 8: Memory-performance curves for a binary 64^3 volume, when (blue) solving the global combinatorial optimization problem of parameter trees, and when (green) utilizing parameter trees returned by our heuristic. The x-axis shows number of fetch operations. The y-axis shows memory consumption. Black lines indicate lower and upper bounds for the memory consumption of SVT representations.

7.1. Quality of the heuristic

We now evaluate how closely the parameter trees returned by the heuristic match the optimal parameter trees. The optimal memory-performance curve for a binary 64^3 volume is shown by the blue curve in Fig. 8. A point (x, y) on the curve indicates that there exists an optimal parameter tree with x fetches and y bits of memory consumption, i.e., if memory is constrained to y bits, our hierarchical data structure principally allows to reduce the number of required fetches to x , and vice versa. The green curve indicates the characteristic of our proposed heuristic. By measuring the shift in x -direction, one can determine the fetch operation overhead of the heuristic for a fixed memory threshold. Compared to the optimal solution, roughly 1.5 times the optimal amount of fetches are required. Note that due to the logarithmic scale of the x -axis, a shift translates to a factor instead of an offset. Vice versa, measuring the shift in y -direction yields the memory overhead of the heuristic, assuming a fixed budget of fetches. Here we can clearly see that the heuristic performs quite well except for the number of 4 fetches, where memory consumption is increased by 25%.

Note that although the memory-performance curve given by the heuristic is not guaranteed to be monotone, it shows a clear falling trend. Thus, when using the bisection method w.r.t. λ to achieve a certain memory threshold, close to perfect results can be expected. We are also confident that the heuristic has not been manually overfitted to the validation scenario. In the design phase, optimal parameter trees for various 1D to 4D arrays with at most 9,000 elements were investigated, while the volume used in the evaluation contains 262,144 elements. On the other hand, due to this we cannot ensure that the results of the evaluation generalize to GB-scale arrays.

7.2. Comparative study

In this study, we compare the SVT representations found by our heuristic to the approaches proposed by Ehsan et al. [ECRMM15],

Table 1: Performance statistics for various SVT representations. Each group of three rows contains results (memory consumption, fetch count) for different approaches using the same volume. For each volume, theoretical lower and upper bounds for SVT representations are given. The **Reference** column shows results for the reference approaches by others. The **Ours** columns show results and timings for parameter tree computation for our SVT representations under varying constraints. Either memory or fetch count is constrained according to the reference in the same row. Note that the non-constrained quantities (in **bold**) are significantly lower than the corresponding reference quantities.

Volume	Reference			Ours (Memory \leq Ref. Memory)			Ours (Fetch \leq Ref. Fetch)		
	Name	Memory	Fetch	Memory	Fetch	Timing	Memory	Fetch	Timing
$256 \times 256 \times 256$ Size: 2MB SVT: 50MB	Ehsan	36.6MB	8	35.1MB	3	20ms	14.9MB	8	41ms
	Part. SVT	32MB	512	27.0MB	4	25ms	3.9MB	476	8.0s
	Fen. Tree	8.0MB	512	8.0MB	38	8.5s	3.9MB	476	8.1s
$1024 \times 1024 \times 1024$ Size: 128MB SVT: 3.9GB	Ehsan	2.8GB	8	2.7GB	3	23ms	1.2GB	8	52ms
	Part. SVT	2GB	32K	1.5GB	5	147ms	170.1MB	27.5K	31.4s
	Fen. Tree	511.6MB	1000	511.6MB	40	6.9s	252.7MB	952	20.5s
$2048 \times 2048 \times 2048$ Size: 1GB SVT: 34GB	Ehsan	24.3GB	8	24.2GB	3	26ms	10.3GB	8	56ms
	Part. SVT	16GB	256K	12.7GB	5	161ms	1.2GB	255.8K	33.5s
	Fen. Tree	4.0GB	1331	4.0GB	39	9.7s	1.9GB	1294	17.3s
$641 \times 9601 \times 9601$ Size: 6.9GB SVT: 247.6GB	Ehsan	176.6GB	8	168.1GB	3	30ms	71.2GB	8	51ms
	Part. SVT	110.1GB	1.8M	86.7GB	5	157ms	7.5GB	1.7M	40.4s
	Fen. Tree	27.5GB	1521	27.0GB	46	11.8s	11.8GB	1468	34.6s
$8192 \times 8192 \times 8192$ Size: 64GB SVT: 2.5TB	Ehsan	1.8TB	8	1.8TB	3	51ms	725.5GB	8	40ms
	Part. SVT	1TB	16M	892.8GB	5	100ms	67.9GB	10.6M	39.1s
	Fen. Tree	256.0GB	2197	252.7GB	46	10.7s	114.4GB	2188	45.4s

Mishra [Mis13], Zellmann et al. [ZSL18] and Urschler et al. [UBD13]. Table 2 summarizes the qualitative features supported by the varying approaches. For a quantitative analysis, we manually compute the memory consumption as well as the number of fetch operations for all reference methods except for Urschler et al., which will be covered later. We use binary volumes of shape 256^3 and $1K^3$, to reproduce the results of the alternatives from other works. To demonstrate the scalability of our approach, additional results using large scale binary datasets from $2K^3$ to $8K^3$ are presented. The results of these experiments are given in Table 1. They generalize to arrays with elements of arbitrary precision p , by adding an offset of $(p-1) \cdot n$ to all memory footprints of both our and reference methods.

It can be seen that our proposed heuristic performs significantly better than any other approach relying on intermediate sum computation. In all cases, we can achieve an improvement of more than a factor of 2.5 in memory consumption or number of fetch operations while matching the budget regarding the respective other quantity. Notably, while it seems that the approach by Ehsan is in all scenarios only this factor behind us, it cannot reduce the memory consumption any further. Thus, where Ehsan requires 1.8TB, our SVT variant can go down as low as 64GB. In comparison to Partial SVTs, we can trade almost all fetch operations for the memory requirement of 67.9GB, and can reduce the memory requirement about more than 90% at the same number of fetch operations. Compared to the 3D Fenwick Tree, our SVT representation requires only 2% of the number of fetch operations at similar memory consumption.

It is fair to say, however, that the improvements over Partial SVTs with respect to memory requirement become less significant with increasing sparsity of the volume. Partial SVTs first split the

3D array into subarrays of size 32^3 , so that empty subarrays can be pruned and do not need to be stored. Even though the so generated sparse structure requires a certain overhead to encode the sparsity information, it is likely that at extreme sparsity levels the Partial SVTs become competitive with respect to memory consumption. It is, on the other hand, not the case that the number of fetch operations decreases similarly, since indirect memory access operations are required to step along the sparse encoding.

Another comparison we perform is against MEIV of Urschler et al. [UBD13], by reusing the array shapes and precision of the datasets used in the authors' work. We set the lowest achieved memory consumption achieved by MEIV as fixed memory budget and compute the parameter tree according to Sec. 6. For the *smallRandomVolume* dataset of shape 512^3 and a maximal possible entry of 1023 we require 6 fetch operations and 311MB of memory (compared to 319MB by MEIV). For the *largeRandomVolume* dataset of shape $1K^3$ and a maximal possible entry of 512 we again require 6 fetch operations, but 2491MB of memory (compared to 2544MB by MEIV). In the case of the *realCTVolume* dataset with shape $512 \times 512 \times 476$ and a maximal possible entry of 1023, MEIV achieves roughly 350MB of memory consumption. We achieve 304MB while requiring 5 fetch operations. Clearly, we achieve results of equal quality compared to MEIV without having its limitations as described in Sec. 2.

7.3. Meteorological use case

In meteorological and climatological research, historical weather data such as the publicly available ERA5 data set [HBB*20] containing global, atmospheric reanalysis data is often analyzed using statistical measures like mean and variance over spatial subdomains and time intervals. These measures indicate trends and can

Table 2: Properties of different SVT representations where n is the number of data values in the volume. Besides memory consumption, we show runtime-complexities for reconstructing an arbitrary partial sum, reconstructing an actual data value, single-threaded construction of the data structure, and updating the representation after a single data value is changed. Further, we indicate if memory consumption can be predicted before constructing the representation, if construction is straightforwardly parallelizable, and if representations can be modified to exploit sparsity in datasets. For our approach, we present lower and upper complexity bounds for all achievable representations. (*) Average runtime-complexity is shown. Worst Case complexity is $\mathcal{O}(\log^3 n)$. (**) Our approach reconstructs data values with the same amount of fetches as required for partial sum reconstruction by regarding data values as partial sums over hyperboxes of size 1. (***) Update performance heavily depends on the actual parameter tree (see Sec. 5.3).

	Verbatim	SVT	Ehsan	Part. SVT	Fen. Tree	MEIV	Ours
Memory consumption	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$ large const.	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$ better in practice	$\mathcal{O}(n) - \mathcal{O}(n \log n)$
Read partial sum	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$ small const.	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(1)$	$\mathcal{O}(1) - \mathcal{O}(n)$
Read data value	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$ *	$\mathcal{O}(1)$	$\mathcal{O}(1) - \mathcal{O}(n)$ **
Construction time	–	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$ large const.	$\mathcal{O}(n)$
Data value update	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(n)$	$\mathcal{O}(1) - \mathcal{O}(n)$ ***
Predictable memory consumption	✓	✓	✓	✓	✓	x	✓
Parallelizable construction	–	✓	✓	✓	✓	✓	✓
Can exploit sparsity	✓	x	x	✓	x	x	?

be used to reveal correlations between observed physical quantities at different sub-domains and times.

As a use case, we utilize the 2m temperature field of the ERA5 hourly data on single levels from 1979 to 2020 [HBB*18]. Data is rescaled to 8bit integers according to Sec. 4 and hourly data is aggregated per day, yielding an 8bit precision dataset of size $1440 \times 721 \times 15404$. We use our parameterized SVT representation to plot the mean temperature progressions over different interactively selected, spatial sub-domains of 150×50 grid points (roughly matching the extend of the Sahara) in a line plot. Assuming a horizontal resolution of 200 pixels, we partition the user chosen time range in 200 equally large intervals and compute one aggregate for each pixel. When viewing the whole temporal domain, each aggregate thus describes a sub-domain of $150 \times 50 \times 77$ grid points and requires 577.500 fetches from the verbatim dataset of size 14.9GB. A SVT requires only 8 fetch operations but 78.2GB of memory. In contrast, by employing our parameterized SVT representation using 22.8GB of memory, we can still perform the computation of mean values per any sub-domain using $31 \cdot 8$ fetch operations, facilitating an interactive visual analysis of arbitrary sub-regions.

8. Conclusion and future work

We have proposed a versatile data structure and heuristic for generating SVT representations that can flexibly trade speed for memory. Hence, SVT representations that are specifically built for a fixed memory or compute budget can be utilized. In a number of experiments on large scale datasets we have compared the resulting SVT representations to those by others, and we have demonstrated significantly reduced memory consumption at similar decoding performance, or vice versa.

In the future, we intend to address the following issues: Firstly, we will engineer cache-aware and/or GPU-accelerated encoding and decoding schemes, so that a) decoding can further benefit from massive parallelism and b) encoding can be realised in timings similar to state-of-the-art SAT encoding. [CWT*18, EFT*18, HSO07] Secondly, we will apply our approach to build spatial acceleration structures such as BVHs for large-scale mesh or volume datasets. Further, we plan to efficiently build implicit BVH structures for DVR that optimize for low variance in density per bounding volume. By using the technique described by Phan et al. [PSCL12], our technique allows to compute variances in constant time without running out of memory. Note that memory efficient SVT implementations are especially important in this regard, because the SVT that is used for computing second order moments is created from an input of double precision than the dataset. Third, we will investigate potential optimizations for sparse data. Here we will address how much memory used by our data structure can be saved by pruning empty subarrays, and whether the heuristic can be adapted to respect empty regions in the dataset.

Further, we plan to extend our approach to nominal data, that is, computing SVTs of histograms instead of scalar entries. Applications are in any research field processing segmented volumes such as neuroscience or material science. For instance, Al-Thelaya et al. [ATAS21] perform sub-volume queries over nominal data to enable real-time computation of local histograms over user selected regions. However, due to arranging histograms in a Mip Map structure, their approach requires an additional footprint assembly step that quickly becomes unfeasible if very large regions are selected. By replacing the Mip Map architecture with our SVT scheme, we believe it will be possible to build a sophisticated mixture graph that allows to skip the footprint assembly step entirely.

9. Acknowledgements

Open access funding enabled and organized by Projekt DEAL. [Correction added on 05 November 2021, after first online publication: Projekt Deal funding statement has been added.]

References

- [ATAS21] AL-THELAYA K., AGUS M., SCHNEIDER J.: The Mixture Graph-A Data Structure for Compressing, rendering, and querying segmentation histograms. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 645–655. doi:10.1109/TVCG.2020.3030451. 1, 10
- [Bel08] BELT H. J. W.: Word Length Reduction for the Integral Image. In *15th IEEE International Conference on Image Processing* (2008), pp. 805–808. doi:10.1109/ICIP.2008.4711877. 3
- [BSB10] BHATIA A., SNYDER W. E., BILBRO G.: Stacked Integral Image. In *IEEE International Conference on Robotics and Automation* (2010), pp. 1530–1535. doi:10.1109/ROBOT.2010.5509400. 2
- [BTVG06] BAY H., TUYTELAARS T., VAN GOOL L.: SURF: Speeded Up Robust Features. In *Computer Vision – ECCV 2006* (Berlin, Heidelberg, 2006), Leonardis A., Bischof H., Pinz A., (Eds.), Springer Berlin Heidelberg, pp. 404–417. 2
- [Cro84] CROW F. C.: Summed-Area Tables for Texture Mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1984), SIGGRAPH '84, Association for Computing Machinery, p. 207–212. doi:10.1145/800031.808600. 1, 2
- [CWT*18] CHEN P., WAHIB M., TAKIZAWA S., TAKANO R., MATSUOKA S.: Efficient Algorithms for the Summed Area Tables Primitive on GPUs. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2018), pp. 482–493. doi:10.1109/CLUSTER.2018.00064. 10
- [ECRMM15] EHSAN S., CLARK A. F., REHMAN N. U., McDONALD-MAIER K. D.: Integral Images: Efficient Algorithms for Their Computation and Storage in Resource-Constrained Embedded Vision Systems. *Sensors* 15, 7 (2015), 16804–16830. doi:10.3390/s150716804. 3, 6, 8
- [EFT*18] EMOTO Y., FUNASAKA S., TOKURA H., HONDA T., NAKANO K., ITO Y.: An Optimal Parallel Algorithm for Computing the Summed Area Table on the GPU. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2018), pp. 763–772. doi:10.1109/IPDPSW.2018.00123. 10
- [Fen94] FENWICK P. M.: A New Data Structure for Cumulative Frequency Tables. *Software: Practice and Experience* 24, 3 (1994), 327–336. doi:10.1002/spe.4380240306. 1, 3
- [FLML14] FACCILOLO G., LIMARE N., MEINHARDT-LLOPIS E.: Integral Images for Block Matching. *Image Processing On Line* 4 (2014), 344–369. doi:10.5201/ipol.2014.57. 2
- [GGB06] GRABNER M., GRABNER H., BISCHOF H.: Fast Approximated SIFT. In *Computer Vision – ACCV 2006* (Berlin, Heidelberg, 2006), Narayanan P. J., Nayar S. K., Shum H.-Y., (Eds.), Springer Berlin Heidelberg, pp. 918–927. 2
- [GM19] GANTER D., MANZKE M.: An Analysis of Region Clustered BVH Volume Rendering on GPU. *Computer Graphics Forum* 38, 8 (2019), 13–21. doi:10.1111/cgfg.13756. 2
- [HBB*18] HERSBACH H., BELL B., BERRISFORD P., BIAVATI G., HORÁNYI A., MUÑOZ, SABATER J., NICOLAS J., PEUBEY C., RADU R., ROZUM I., SCHEPERS D., SIMMONS A., SOCI C., DEE D., THÉPAUT J.-N.: ERA5 hourly data on single levels from 1979 to present, 2018. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). Accessed on 08-03-2021. doi:10.24381/cds.adbb2d47. 10
- [HBB*20] HERSBACH H., BELL B., BERRISFORD P., HIRAHARA S., HORÁNYI A., MUÑOZ-SABATER J., NICOLAS J., PEUBEY C., RADU R., SCHEPERS D., SIMMONS A., SOCI C., ABDALLA S., ABELLAN X., BALSAMO G., BECHTOLD P., BIAVATI G., BIDLOT J., BONAVITA M., DE CHIARA G., DAHLGREN P., DEE D., DIAMANTAKIS M., DRAGANI R., FLEMMING J., FORBES R., FUENTES M., GEER A., HAIMBERGER L., HEALY S., HOGAN R. J., HÓLM E., JANISKOVÁ M., KEELEY S., LALOYAX P., LOPEZ P., LUPU C., RADNOTI G., DE ROSNAY P., ROZUM I., VAMBORG F., VILLAUME S., THÉPAUT J.-N.: The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society* 146, 730 (2020), 1999–2049. URL: <https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3803>, arXiv:<https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.3803>, doi:<https://doi.org/10.1002/qj.3803>. 9
- [Hec86] HECKBERT P. S.: Filtering by Repeated Integration. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 315–321. doi:10.1145/15886.15921. 2
- [HHS06] HAVRAN V., HERZOG R., SEIDEL H.: On the Fast Construction of Spatial Hierarchies for Ray Tracing. In *IEEE Symposium on Interactive Ray Tracing* (2006), pp. 71–80. doi:10.1109/RT.2006.280217. 1, 2
- [HPD08] HUSSEIN M., PORIKLI F., DAVIS L.: Kernel Integral Images: A Framework for Fast Non-Uniform Filtering. In *IEEE Conference on Computer Vision and Pattern Recognition* (2008), pp. 1–8. doi:10.1109/CVPR.2008.4587641. 2
- [HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast Summed-Area Table Generation and its Applications. *Computer Graphics Forum* 24, 3 (2005), 547–555. doi:10.1111/j.1467-8659.2005.00880.x. 2
- [HSO07] HARRIS M., SENGUPTA S., OWENS J. D.: Parallel prefix sum (scan) with CUDA. *GPU gems* 3, 39 (2007), 851–876. 10
- [Mis13] MISHRA P.: A New Algorithm for Updating and Querying Sub-arrays of Multidimensional Arrays. *arXiv* (2013). arXiv:1311.6093v6. 1, 3, 8, 9
- [Por05] PORIKLI F.: Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005), vol. 1, pp. 829–836 vol. 1. doi:10.1109/CVPR.2005.188. 2
- [PSCL12] PHAN T., SOHONI S., CHANDLER D. M., LARSON E. C.: Performance-Analysis-Based Acceleration of Image Quality assessment. In *IEEE Southwest Symposium on Image Analysis and Interpretation* (2012), pp. 81–84. doi:10.1109/SSIAI.2012.6202458. 2, 10
- [SKB08] SHAFAIT F., KEYSERS D., BREUEL T. M.: Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images. In *Document Recognition and Retrieval XV* (2008), Yanikoglu B. A., Berkner K., (Eds.), vol. 6815, International Society for Optics and Photonics, SPIE, pp. 317–322. doi:10.1117/12.767755. 2
- [SR17] SCHNEIDER J., RAUTEK P.: A Versatile and Efficient GPU Data Structure for Spatial Indexing. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 911–920. doi:10.1109/TVCG.2016.2599043. 1, 3
- [UBD13] URSCHLER M., BORNIK A., DONOSER M.: Memory Efficient 3D Integral Volumes. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops* (June 2013). 3, 8, 9
- [VJ04] VIOLA P., JONES M. J.: Robust Real-Time Face Detection. *International Journal of Computer Vision* 57, 2 (2004), 137–154. 2
- [VMD08] VIDAL V., MEI X., DECAUDIN P.: Simple Empty-Space Removal for Interactive Volume Rendering. *Journal of Graphics Tools* 13, 2 (2008), 21–36. doi:10.1080/2151237X.2008.10129258. 1, 2
- [Wil83] WILLIAMS L.: Pyramidal Parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1983), SIGGRAPH '83, Association for

Computing Machinery, p. 1–11. doi:10.1145/800059.801126.
2

[ZSL18] ZELLMANN S., SCHULZE J. P., LANG U.: Rapid kd Tree Construction for Sparse Volume Data. In *Proceedings of the Symposium on Parallel Graphics and Visualization* (Goslar, DEU, 2018), EGPGV '18, Eurographics Association, p. 69–77. 3, 8, 9

Supplementary Material to the Publication Parameterized Splitting of Summed Volume Tables

Christian Reinbold  and Rüdiger Westermann 

Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany

1. Details of the Heuristic

In our work we did not elaborate on two technical aspects of the heuristic, namely the interpolation scheme to derive the subarray size z and the ratio correlation to partition λ into λ_a and λ_s . We now make up for it so that the heuristic can be reimplemented accurately. Further, we present pseudo code to ease implementation.

1.1. Subarray size

In order to derive the heuristic, we computed sets of optimal parameter trees via an expensive Branch-and-Bound approach for arrays of small shape. When investigating these, we noticed that the subarray size parameter at the root node correlates with the fetch estimate computed for a parameter tree. For instance in Fig 1, all optimal parameter trees for a binary 2D array of shape 64^2 are represented via points (x, y) , with x being the fetch estimate and y being the subarray size parameter at the root node. For all investigated array shapes, most points roughly follow a logarithmic curve. We found out that the function

$$f(\lambda) := \frac{\log_2(2 \cdot \lambda / (0.35 \cdot n_k) + 1)}{\log_2(2|n|/n_k + 1)},$$

where n is the shape of the input array and k is the split dimension, allows to interpolate between the smallest and largest occurring subarray size z . Clearly, $z \geq 1$. Since one can have up to three subarrays when having only one split position (keep in mind the conjugate trick, see Sec. 4.2), distributed alignment yields maximal subarray sizes of $\lceil (n_k - 1)/3 \rceil$. Setting $z = (1 - f(\lambda)) \cdot 1 + f(\lambda) \cdot \lceil (n_k - 1)/3 \rceil$ yields the green curve of Fig. 1. After rounding to the nearest subarray shape that can actually arise in distributed aligned splits, we obtain our final estimate of z . The orange curve of Fig. 1 depicts which subarray size is chosen for which fetch estimate. Note that it roughly follows the distribution of optimal parameter trees.

1.2. Finding fetch estimates for recursion

Again, by looking at optimal parameter trees, we made the observation that the fetch estimate of the first aggregate subtree can be approximated as well. This is easily seen by considering some ratios. Let T be an optimal parameter tree for an input array of shape

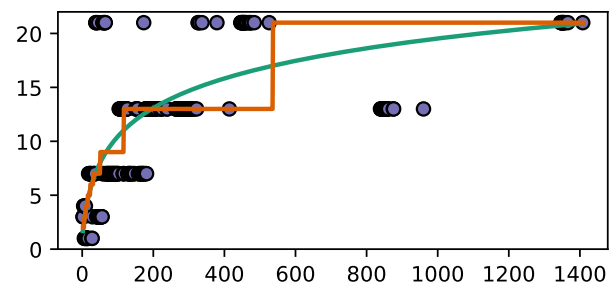


Figure 1: Scatterplot relating fetch estimates (x -axis) to the subarray size parameter at the root node (y -axis) for all optimal parameter trees for a binary 2D array of shape 64^2 . Blue points represent optimal parameter trees. Curves indicate the heuristic used for estimating subarray size from the fetch estimate (**green**) before and (**orange**) after discretization.

n and let T_a be the aggregate subtree of T describing the split hierarchy of an aggregate array of shape $n^{(a)}$. We noticed that

$$\frac{\text{FETCH}(T_a)}{\text{FETCH}(T)} \approx 2 \cdot \frac{|n^{(a)}|}{|n|}.$$

In Fig. 2, optimal parameter trees for a binary 64^2 -shaped array are represented via points (x, y) where x represents the ratio $|n^{(a)}|/|n|$ and y the ratio $\text{FETCH}(T_a)/\text{FETCH}(T)$. The orange curve hints at the conjectured linear correlation.

Since the control parameter λ of the heuristic represents a threshold for the fetch estimate, it is reasonable that the control parameter λ_a is chosen such that the conjectured correlation holds true. This can be achieved by setting $\lambda_a := 2 \cdot \ell / n_k \cdot \lambda$, where ℓ is the number of split positions. It can be computed from the subarray size z via $\ell = \lceil (n_k - z) / (2 \cdot z + 1) \rceil$. As a last step, we round λ_a to the nearest integral number and clamp it to the interval $[1, \min(|n^{(a)}|, \lambda - 1)]$. This ensures a) that λ_a cannot surpass the maximal amount of $|n^{(a)}|$ fetches for the aggregate array, and b) that both λ_a and the number of fetches $\lambda_s := \lambda - \lambda_a$ that remain for the subarrays is at least one. Algorithm 1 depicts the pseudo code of the heuristic after incorporating the considerations of this section.

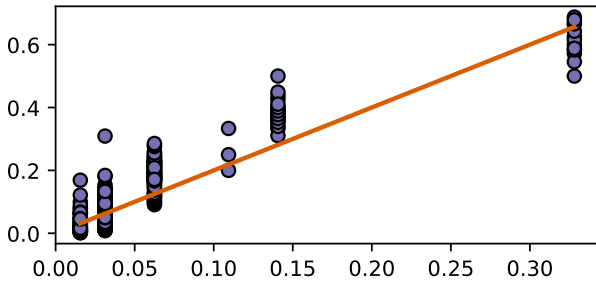


Figure 2: Scatterplot relating the ratio $|n^{(a)}|/|n|$ (x-axis) to the ratio $\text{FETCH}(T_a)/\text{FETCH}(T)$ (y-axis) for all optimal parameter trees for a binary 2D array of shape 64^2 . Blue points represent optimal parameter trees. The orange curve indicates the conjectured linear correlation between both ratios.

2. A tighter bound for fetch operations

Although the recursive formula for estimating the fetch operations in Sec. 5.2 is a useful tool for deriving the heuristic, it is not the actual number we are interested in. Instead, we wish to know the maximal number of fetch operations we require when querying an arbitrary prefix sum. Eq. (3) overestimates this number since it assumes that one has to query a prefix sum from the full aggregate array and a full subarray simultaneously to get the recursion going. This is not quite true in the scenario of Fig. 3 which—for demonstration purposes—does not utilize the trick of conjugating subarrays. Here, we have $\text{FETCH}(T_a) = \text{FETCH}(T_{s_1}) = 2$ and $\text{FETCH}(T_{s_2}) = 1$. $\text{FETCH}(T)$ evaluates to 4. In the estimate formulation it is assumed that whenever we query into a subarray of size 2, we have to query both aggregates as well. However, we only have to query aggregates at *preceding* split positions. Thus, the second aggregate entry never is fetched as it is not followed by a subarray of size 2. Consequently, querying into one of the first two subarrays requires at most 3 fetches. It is only in the case of querying into the last subarray of size 1 that we require both aggregate entries. But then querying the smaller subarray requires only one fetch, summing up to 3 as well. If we query into an aggregate array instead of a subarray, one would require at most 2 fetches. As a result, one requires at most 3 fetch operations instead of 4 as indicated by the fetch estimate.

In order to obtain a tighter bound, we have to make sure that we only pair fetch operations of a subarray with the fetch operations that are required to evaluate prefix sums of the aggregate array up to the *preceding* split position; or up to the *subsequent* split position in the case of a conjugated subarrays. To resolve this issue, we introduce a more complex recursion formula $P(T, v)$ that takes a parameter tree T and a corner $v \in \mathbb{N}^d$, and then returns an upper bound for fetch operations required to retrieve any prefix sum over a subarray contained in the *box* of v , that is the volume spanned by the origin and the corner v . Any box spanned by the origin and a point in the box of v is called a *subbox* of v .

Algorithm 1: heuristic for approximately solving the optimal parameter tree problem.

```

1 Function  $\mathcal{H}$ (control parameter  $\lambda$ , array shape  $n \in \mathbb{N}^d$ ):
   Output: Parameter tree  $T$  with at most  $\lambda$  fetches for
         array of shape  $n$ 
2    $T \leftarrow$  new Node();
   /* Covering leaf cases */
3   if  $\lambda = 1$  then // SVT block
4     attach parameter  $I = \{1, 2, \dots, d\}$  to  $T$ ;
5     return  $T$ ;
6   else if  $\lambda \geq |n|$  then // verbatim block
7     attach parameter  $I = \emptyset$  to  $T$ ;
8     return  $T$ ;
9   end
   /* Defining split parameters */
10   $k \leftarrow \arg \max_i (n_i)$ ;
11   $f \leftarrow \log_2(2 \cdot \lambda / (0.35n_k + 1)) / \log_2(2|n|/n_k + 1)$ ;
12   $z_{\text{continuous}} \leftarrow (1 - f) \cdot 1 + f \cdot \lceil (n_k - 1) / 3 \rceil$ ;
13   $z \leftarrow$  round  $z_{\text{continuous}}$  to the nearest subarray size that
         can occur with distributed aligned splits along a split
         dimension of length  $n_k$ ;
14  attach parameters  $k, z$  to  $T$ ;
   /* Creating subtrees via recursion */
15   $\ell \leftarrow \lceil (n_k - z) / (2 \cdot z + 1) \rceil$ ;
16   $n^{(a)} \leftarrow n|_{k=\ell}$ ;
17   $n^{(s_1)} \leftarrow n|_{k=z}$ ;
18   $n^{(s_2)} \leftarrow n|_{k=z-1}$ ;
19   $\lambda_a \leftarrow \text{round}(2 \cdot \ell / n_k \cdot \lambda)$ ;
20  Clamp  $\lambda_a$  to  $[1, \min(|n^{(a)}|, \lambda - 1)]$ ;
21   $\lambda_s \leftarrow \lambda - \lambda_a$ ;
22   $T_a \leftarrow \mathcal{H}(\lambda_a, n^{(a)})$ ;
23   $T_{s_1} \leftarrow \mathcal{H}(\lambda_s, n^{(s_1)})$ ;
24   $T_{s_2} \leftarrow \mathcal{H}(\lambda_s, n^{(s_2)})$ ;
25  Attach  $T_a, T_{s_1}$  and  $T_{s_2}$  to  $T$ ;
26  return  $T$ ;

```

2.1. Leaf nodes

If T consists of a single leaf node, evaluation of $P(T, v)$ is straightforward. We just have to compute the number of entries along the axes for which values have not been cumulated before storing them. Hence, we read off the I parameter of the leaf node and set

$$P(T, v) = \prod_{i \in \{1, \dots, d\} \setminus I} v_i.$$

2.2. Split nodes

If the root of T is an internal node, computations become more involved. We differentiate between upper bounds of fetches for each subarray shape $n^{(s)}$ arising in the split operation via an operation $P_s(T, v, n^{(s)})$. Similarly, we define a function $P_a(T, v)$ that returns an upper bound of fetches if one queries into an aggregate array.

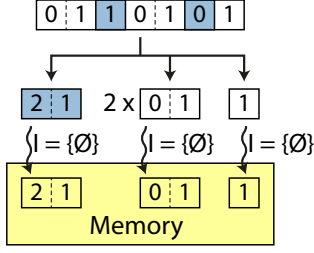


Figure 3: Decomposition of a 1D array of size 7 according to a parameter tree describing a single split with subarray size 2. All entries at leaf nodes are stored verbatim.

Clearly, $P(T, v)$ can be computed by taking the maximum over the value of P_a and the value of P_s for each subarray shape. In the scenario of Fig. 3, P_s evaluates to 3 for both subarray shapes and P_a evaluates to 2. Thus, P evaluates to the maximum of 3 as well.

From now on let us assume that T is an internal node with an aggregate subtree T_a and one or more subarray subtrees T_s^i , $i \in \{1, 2, \dots, \#\text{subarray shapes}\}$. The split dimension of T is denoted by k . Further we assume a fixed array F of shape n that splits into an aggregate subarray F_a of shape n_a and subarrays F_s^i of different shapes n_s^i such that the subarray subtree T_s^i relates to shape n_s^i .

2.2.1. Query to aggregate arrays

$P_a(T, v)$ is computed as follows: First, the index of the last split position contained in the box of v is computed. In the notation of Sec. 4 of the paper this is $i = \max(\{m \mid c_m \leq v_k\} \cup \{0\})$. By Eq. (1), the box of $v \mid_{k=i}$ resembles the region in F_a that has to be queried in order to compute a prefix sum of a subbox of v in F . Since we do query into an aggregate array, the subarray offset j of Eq. (1) and thus the number of fetches to evaluate the second summand in Eq. (1) always is 0. By the definition of P , we can set

$$P_a(T, v) = P(T_a, v \mid_{k=i}).$$

Note that P now depends on P_a , which itself depends on P again, but invoked on a subtree. Hence, we have setup a recursion formula.

Now we address the evaluation of $P_s(T, v, n_s^i)$ for a subarray shape n_s^i . We simplify notation by fixing the i for which P_s is evaluated and leave out the superscript, that is $F_s \hat{=} F_s^i$, $n_s \hat{=} n_s^i$ and so on. A subarray \mathcal{S} is called *complete*, if all its entries projected to the split dimension are contained in the box of v . In other words, \mathcal{S} is complete if and only if for each voxel w covered by \mathcal{S} it holds that $w_k \leq v_k$.

2.2.2. Query to complete subarrays

First, we consider the case of querying into a *complete* subarray of shape n_s . W.l.o.g. we can assume that the queried subarray is the "last" complete subarray $\mathcal{S}_{\text{last}}$ which is most distant from the origin and thus has the highest number of preceding split positions. If we would query into another subarray \mathcal{S} , we can query into $\mathcal{S}_{\text{last}}$ instead and would obtain an upper bound at least as high as when

querying into \mathcal{S} . The upper bound of fetches for the second summand of Eq. (1) remains the same, and the upper bound of fetches for the first summand can only become bigger to due more preceding split positions. Now, we set i to the index of the split position preceding $\mathcal{S}_{\text{last}}$. If $\mathcal{S}_{\text{last}}$ is *not* going to be conjugated, we compute

$$p_{\text{complete}} = P(T_a, v \mid_{k=i}) + P(T_s, v \mid_{k=j}),$$

where $j := (n_s)_k$ is the subarray size along the split dimension. If $\mathcal{S}_{\text{last}}$ is going to be conjugated, Eq. (2) instead of Eq. (1) is used for prefix sum computation, which is why we then have

$$p_{\text{complete}} = P(T_a, v \mid_{k=i+1}) + P(T_a, v \mid_{k=j})$$

instead. Note that i is incremented by one.

2.2.3. Query to incomplete subarrays

Second, we discuss the case of querying into an incomplete subarray of shape n_s , if there is any. Clearly, there can be only one of them and it has to follow the last complete subarray $\mathcal{S}_{\text{last}}$. Let us call it \mathcal{S}_{inc} and, again, denote by i the index of its preceding split position. In comparison to the complete case, we cannot set $j := (n_s)_k$ anymore, but have to keep in mind that we query only those voxels of \mathcal{S}_{inc} that are also contained in the box of v . Hence, we set j to the size along the split dimension obtained after intersecting \mathcal{S}_{inc} with the box of v . If \mathcal{S}_{inc} is *not* going to be conjugated, then it holds that

$$p_{\text{incomplete}} = P(T_a, v \mid_{k=i}) + P(T_s, v \mid_{k=j}).$$

If, on the other hand, \mathcal{S}_{inc} is going to be used as a conjugated subarray in Eq. (2), we have to keep in mind that even if the width j is only one, we have to sum over the whole subarray due to flipping. Hence, then we have

$$p_{\text{incomplete}} = P(T_a, v \mid_{k=i+1}) + P(T_a, v \mid_{k=(n_s)_k}).$$

If there exists no incomplete subarrays at all, we set $p_{\text{incomplete}} = 0$. With both complete and incomplete subarrays covered, we can compute P_s via

$$P_s(T, v, n_s) = \max(p_{\text{complete}}, p_{\text{incomplete}}).$$

2.2.4. Performance considerations

This concludes the formulation of the recursion formula. At this point, formally proving its upper bound property is a mere exercise in precise mathematical formulation and rephrasing of the sections above in a definition-theorem-proof style. However, we still have to improve performance. If j is the number of different subarray shapes (in our case $j = 2$), P has to invoke itself $2j + 1$ times with the aggregate subtree and $2j$ times with a subarray subtree. Since this factors multiply for each level of the tree, computation times quickly get out of hand. Performance can be improved significantly by introducing a cache storing results of previous evaluations of P . In doing so, the algorithm even can be used in the bisection method that tweaks the control parameter λ of the heuristic. Timings given in Table 2 include the costs for repeatedly computing the upper bound as proposed here. Further, note that performance does not depend on the actual array size because subarray shapes as well as indices i, j can be derived from split parameters and the input array shape in constant time. Pseudo code for the complete implementation is shown in Algorithm 2.

Algorithm 2: Computation of a tighter bound for fetch operations respecting queries into subboxes.

Input: Parameter tree T ,
 index $v \in \mathbb{N}^d$ into the input array,
 (Optional) cache datastructure \mathcal{C}

Output: Upper bound $P(T, v)$ for fetch operations required to retrieve prefix sums over any subbox of v when using the SVT representation defined by T

```

1 Initialize empty cache  $\mathcal{C}$  if not provided;
2 if  $(T, v) \in \mathcal{C}$  then
3   | return cached result  $P(T, v)$ ;
4 end

5 if root node of  $T$  is leaf then
6   | Read off  $I$  parameter from root of  $T$ ;
7   | result  $\leftarrow \prod_{i \in \{1, \dots, d\}} I v_i$ ;
8   | Insert result into  $\mathcal{C}$  at position  $(T, v)$ ;
9   | return result;
10 else
11   | Read off split dimension index  $k$  from root of  $T$ ;
12   |  $i \leftarrow$  index of last split position contained in the box of  $v$ ;
13   | upper  $\leftarrow P(T_a, v |_{k=i}, \mathcal{C})$ ;
14   | foreach subarray subtree  $T_s$  of  $T$  do
15     |  $n_s \leftarrow$  array shape associated to  $T_s$ ;
16     | Find last complete subarray  $\mathcal{S}_{\text{last}}$ ;
17     |  $i \leftarrow$  index of last split position preceding  $\mathcal{S}_{\text{last}}$ ;
18     | if  $\mathcal{S}_{\text{last}}$  is conjugated then
19       |  $p \leftarrow P(T_a, v |_{k=i+1}, \mathcal{C}) + P(T_a, v |_{k=(n_s)_k}, \mathcal{C})$ ;
20     | else
21       |  $p \leftarrow P(T_a, v |_{k=i}, \mathcal{C}) + P(T_a, v |_{k=(n_s)_k}, \mathcal{C})$ ;
22     | end
23     | upper  $\leftarrow \max(\text{upper}, p)$ ;
24     | Search for incomplete subarray  $\mathcal{S}_{\text{inc}}$ ;
25     | if  $\mathcal{S}_{\text{inc}}$  exists then
26       |  $i \leftarrow$  index of last split position preceding  $\mathcal{S}_{\text{inc}}$ ;
27       |  $j \leftarrow$  size along split dimension  $k$  of intersection
28       |   of  $\mathcal{S}_{\text{inc}}$  and the box of  $v$ ;
29       | if  $\mathcal{S}_{\text{inc}}$  is conjugated then
30         |  $p \leftarrow P(T_a, v |_{k=i+1}, \mathcal{C}) +$ 
31         |    $P(T_s, v |_{k=(n_s)_k}, \mathcal{C})$ ;
32       | else
33         |  $p \leftarrow P(T_a, v |_{k=i}, \mathcal{C}) + P(T_s, v |_{k=j}, \mathcal{C})$ ;
34       | end
35       | upper  $\leftarrow \max(\text{upper}, p)$ ;
36     | end
37   | Insert upper into  $\mathcal{C}$  at position  $(T, v)$ ;
38   | return upper;
39 end

```

3. SVT representations store partial sums

When discussing construction costs of the proposed SVT representations in Sec. 5.4, we assume that each value stored in memory is a partial sum of the input array F and thus can be sampled from a SVT of F . This claim is intuitive insofar as values of aggregate

arrays are obtained by summing consecutive values of the array being split. In the following, we give a formal proof of this claim in Theorem 3.6. However, we require some additional notation first.

Definition 3.1. Let $c = (c_1, c_2, \dots, c_n)$ be a sequence of n natural numbers. We denote by $|c|$ its length n . For another sequence d of natural numbers such that $d_i \leq n$ for all $i \in \mathbb{N}_{\leq |d|}$, the concatenation $c \circ d$ of c and d is the sequence of length $|d|$ given by $(c \circ d)_i = c_{d_i}$. For $m \in \mathbb{N}_{< |c|}$, the sequence $c \ll m$ of length $|c| - m$ is given by $(c \ll m)_i = c_{i+m}$.

Definition 3.2. Let c be a monotone sequence of natural numbers. The range of c at index $i \in \mathbb{N}_{\leq |c|-1}$ is

$$\text{range}(c, i) = \begin{cases} \{n \in \mathbb{N} \mid c_i < n \leq c_{i+1}\} & \text{if } c \text{ is increasing} \\ \{n \in \mathbb{N} \mid c_{i+1} < n \leq c_i\} & \text{if } c \text{ is decreasing.} \end{cases}$$

Lemma 3.3. Let c be a monotone sequence of natural numbers and d be an **increasing** sequence of natural numbers such that $c \circ d$ is defined. Then for all $i \in \mathbb{N}_{\leq |d|}$ and $m \in \mathbb{N}_0$ it holds that

$$\bigcup_{j=d_i}^{d_{i+1}-1} \text{range}(c, j) = \text{range}(c \circ d, i)$$

$$\text{range}(c, i+m) = \text{range}(c \ll m, i)$$

Proof. Follows immediately from the definition of range. \square

Definition 3.4. Let c be a finite (that is $|c| < \infty$) monotone sequence of natural numbers with. The inverse $\neg c$ of c is the monotone sequence given by $(\neg c)_i = c_{|c|-i-1}$.

Lemma 3.5. Let c be a finite monotone sequence of natural numbers. Then for all $i \in \mathbb{N}_{\leq |c|-1}$ it holds that

$$\text{range}(\neg c, i) = \text{range}(c, |c| - i).$$

Proof. Follows immediately from the definition of the inverse and range. \square

Theorem 3.6. Let A be an array of shape $n \in \mathbb{N}^d$ arising in a split hierarchy of the input array F . Then there exist d monotone sequences $a^{(1)}, \dots, a^{(d)}$ of length $|a^{(i)}| = n_i + 1$ such that

$$A[v] = \sum_{v'_i \in \text{range}(a^{(i)}, v_i)} F[v'],$$

where $v \in \mathbb{N}^d$ is any multi index indexing into A . In particular, $A[v]$ is a partial sum of a hyperbox of F that starts at the corner

$$(\min(a_{v_1}^{(1)}, a_{v_1+1}^{(1)}) + 1, \dots, \min(a_{v_d}^{(d)}, a_{v_d+1}^{(d)}) + 1)$$

and ends at

$$(\max(a_{v_1}^{(1)}, a_{v_1+1}^{(1)}), \dots, \max(a_{v_d}^{(d)}, a_{v_d+1}^{(d)}))$$

Proof. The second claim immediately follows from the definition of range. The first claim is proven via induction regarding the number of performed split operations before arriving at A . For the initial case, $A = F$ and the claim holds true by setting $a^{(i)} = (0, 1, \dots, n_i)$.

Next, the inductive step is shown. Assume that A arises by splitting the array B of shape $m \in \mathbb{N}^d$ along split dimension k and split positions

$$0 = c_0 < c_1 < \dots < c_\ell \leq m_k$$

with the array B being split. By the induction hypothesis, B can be written as

$$B[v] = \sum_{v'_i \in \text{range}(b^{(i)}, v_i)} F[v']$$

with appropriate monotone sequences $b^{(i)}$.

Case 1: A is the aggregate array returned by the split process. We define the monotone sequence d of length $\ell + 1$ via $d_m = c_{m-1} + 1$. Then, by Lemma 3.3, we have

$$\begin{aligned} A[v] &= \sum_{i=c_{v_k-1}+1}^{c_{v_k}} B[v \mid_{k=i}] \\ &= \sum_{i=d_{v_k}}^{d_{v_k+1}-1} \sum_{v'_k \in \text{range}(b^{(k)}, i)} \sum_{\substack{v'_j \in \text{range}(d^{(j)}, v_j) \\ j \neq k}} F[v'] \\ &= \sum_{v'_k \in \text{range}(b^{(k)} \circ d, v_k)} \sum_{\substack{v'_j \in \text{range}(d^{(j)}, v_j) \\ j \neq k}} F[v'] \end{aligned}$$

Thus, the claim holds true by setting $a^{(k)} = b^{(k)} \circ d$ and $a^{(j)} = b^{(j)}$ for all $j \neq k$.

Case 2: A is a non-conjugated subarray returned by the split process. Choose i such that c_i is the last split position preceding A . By Lemma 3.3, it is

$$\begin{aligned} A[v] &= B[v \mid_{k=c_i+v_k}] \\ &= \sum_{v'_k \in \text{range}(b^{(k)}, c_i+v_k)} \sum_{\substack{v'_j \in \text{range}(b^{(j)}, v_j) \\ j \neq k}} F[v'] \\ &= \sum_{v'_k \in \text{range}(b^{(k)} \ll c_i, v_k)} \sum_{\substack{v'_j \in \text{range}(b^{(j)}, v_j) \\ j \neq k}} F[v'] \end{aligned}$$


Again, the claim holds true by setting $a^{(k)} = b^{(k)} \ll c_i$ and $a^{(j)} = b^{(j)}$ for all $j \neq k$.

Case 3: A is a conjugated subarray returned by the split process. Choose i such that c_i is the first split position succeeding A . By Lemma 3.3 and 3.5, it is

$$\begin{aligned} A[v] &= B[v \mid_{k=c_i+1-v_k}] \\ &= \sum_{v'_k \in \text{range}(b^{(k)}, c_i+1-v_k)} \sum_{\substack{v'_j \in \text{range}(b^{(j)}, v_j) \\ j \neq k}} F[v'] \\ &= \sum_{v'_k \in \text{range}(-b^{(k)}, m_k - c_i + v_k)} \sum_{\substack{v'_j \in \text{range}(b^{(j)}, v_j) \\ j \neq k}} F[v'] \\ &= \sum_{v'_k \in \text{range}(-b^{(k)} \ll (m_k - c_i), v_k)} \sum_{\substack{v'_j \in \text{range}(b^{(j)}, v_j) \\ j \neq k}} F[v'] \end{aligned}$$

The claim holds true by setting $a^{(k)} = -b^{(k)} \ll (m_k - c_i)$ and $a^{(j)} = b^{(j)}$ for all $j \neq k$. Note that $m_k \geq c_i$. \square

Visualizing the Stability of 2D Point Sets from Dimensionality Reduction Techniques

Christian Reinbold , Alexander Kumpf and Rüdiger Westermann

Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany
{christian.reinbold, alexander.kumpf, westermann}@tum.de

Abstract

We use k -order Voronoi diagrams to assess the stability of k -neighbourhoods in ensembles of 2D point sets, and apply it to analyse the robustness of a dimensionality reduction technique to variations in its input configurations. To measure the stability of k -neighbourhoods over the ensemble, we use cells in the k -order Voronoi diagrams, and consider the smallest coverings of corresponding points in all point sets to identify coherent point subsets with similar neighbourhood relations. We further introduce a pairwise similarity measure for point sets, which is used to select a subset of representative ensemble members via the PageRank algorithm as an indicator of an individual member's value. The stability information is embedded into the k -order Voronoi diagrams of the representative ensemble members to emphasize coherent point subsets and simultaneously indicate how stable they lie together in all point sets. We use the proposed technique for visualizing the robustness of t -distributed stochastic neighbour embedding and multi-dimensional scaling applied to high-dimensional data in neural network layers and multi-parameter cloud simulations.

Keywords: visualization

ACM CCS: • Human-centred computing → Visualization techniques; • Computing methodologies → Dimensionality reduction and manifold learning

1. Introduction

Dimensionality reduction is used to compute a projection of a set of data points in some high-dimensional space into a low-dimensional space, typically two dimensions, in which visual data exploration can then be performed effectively. If the projection preserves relative distances between the initial data points, or at least allows distinguishing near from far neighbours, coherent subsets can be discerned and even computed in an unsupervised manner using clustering techniques in the low-dimensional space.

The results of many dimensionality reduction techniques, however, are subject to randomly selected initial parameters. For instance, multi-dimensional scaling (MDS) [CC00, KW78] sets the initial objects to random positions and then re-adjusts them. Re-adjustment is performed in a physics-inspired way, by pushing objects apart (or together) if two objects are too close to (or too far from) each other. The result of this relaxation depends on tuning parameters to adjust the strength of non-local repulsive forces. Due to this parameter dependency, multiple MDS projections are usually

computed using varying input parameters and the best result configuration is selected. Another frequently used dimensionality reduction technique is t -distributed stochastic neighbour embedding (t -SNE) [vH08]. In t -SNE, perplexity is an adjustable parameter that controls the size of the neighbourhood around each object that the projection attempts to preserve. Besides, t -SNE is often used with random values to initialize its seed configuration, i.e. the initial locations of projected objects which are considered by gradient descent optimization. Therefore, also t -SNE is usually re-run with different input configurations and the most meaningful projection is selected.

Due to these parametrization dependencies, MDS and t -SNE can be run multiple times and yield a different result each time. The variations can be both with respect to the location of projected data points and their neighbourhoods. It is in particular important to find those subsets of points with similar k -neighbourhood, i.e. the k closest points, in all projections. Such stable subsets can be deemed coherent, and they indicate robustness of the dimensionality reduction technique. Analysing the stability of these k -neighbourhoods,

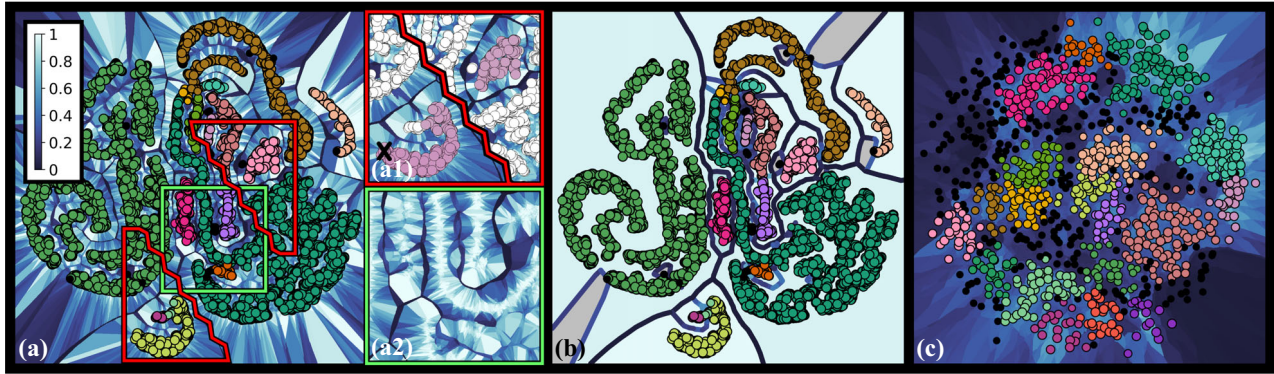


Figure 1: Robustness plots (using 100 projections) of representative projections for t-SNE on 12D parameter vectors of voxels in a cloud dataset (a, b), and multi-dimensional scaling on activations of a convolutional layer of neural classification network (c). (a, c) Voronoi plot reveals structures of high (light blue) to low (dark blue) stability. Point colour indicates groups of similarly stable k -neighbourhoods or outliers. (a1) By picking the marked point, points assigned to the same group of similar k -neighbourhoods in $\geq 70\%$ of all projections are highlighted. (a2) Background Voronoi structure without points. (b) Simplified robustness plot of (a). Grey indicates outlier regions, other colours as in (a).

in context of the projections that are computed by the used dimensionality reduction technique, is the major focus of this work.

1.1. Contribution

We present a visualization technique to analyse the stability of neighbourhood relations in an ensemble of two-dimensional (2D) data points, and we demonstrate its use for assessing the robustness of dimensionality reduction techniques to variations in the input parametrizations. A novel coherence indicator based on k -order Voronoi diagrams measures how well k -neighbourhoods are preserved in multiple ensemble members.

The visual encoding we propose utilizes the partitioning of the projection plane based on k -order Voronoi diagrams. In this way, the spatial distribution of data points is maintained, and stability information of similar neighbourhoods is placed close to each other. This allows accessing in a very intuitive way region of neighbourhoods with similar stability, which cannot be accessed from non-spatial encodings like, e.g. bar charts. Since for each projection, a k -order Voronoi diagram is computed, we select the most representative one via the PageRank score. Voronoi cells are colour-coded according to the stability of enclosed subsets of data points, and the stability information is used to simplify the visual encoding by removing cell boundaries. Remaining cell boundaries are coloured according to how strongly separated the subsets in either cell are.

Our specific contributions are:

- the use of k -order Voronoi diagrams to analyse the stability of neighbourhood relations in an ensemble of 2D point sets;
- a stability measure for k -neighbourhoods in ensembles of 2D point sets, using smallest coverings of points belonging to a k -neighbourhood computed for all ensemble members;
- a visual encoding of the stability of neighbourhood relations, by visualizing major separating ridges in a k -order Voronoi diagram via colours indicating their separation strength.

Figure 1 demonstrates the application of the proposed technique to analyse the robustness of MDS and t-SNE for two different datasets, indicating the specific visual encoding we propose. It provides a general means to analyse the stability of neighbourhood relations in ensembles of 2D point sets. We use this means to shed light on how to determine for a given dimensionality reduction technique whether it can robustly reveal certain structures in the data, yet we do not aim to compare different dimensionality reduction techniques. In particular, we show that the proposed visualizations can be used to find subgroups of objects for which a current projection provides a robust representation of their locality. In a number of examples, including a synthetic dataset, we demonstrate the use of the proposed technique and the information concerning the robustness of dimensionality reduction techniques they provide.

2. Related Work

Our approach takes as input an ensemble of 2D point sets and visualizes which subsets of points are connected to each other, i.e. are neighbours, in all or many ensemble members using a suitable neighbourhood relation. The neighbourhood relation we employ is based on Voronoi diagrams. In visualization, classical one-order Voronoi diagrams have been used to encode properties of points [BD05, NSB04, PFMA06, LA11]. Aupetit [Aup07] discusses the limitations of two-order Voronoi diagrams for visual encoding properties of pairs of points, and suggests an improvement using Segment-Voronoi cells. We use higher order Voronoi diagrams to determine larger contiguous neighbourhoods, and clustering to obtain a meaningful visual encoding, thus resolving some of the limitations pointed out by Aupetit.

Regarding the input to our approach, it falls into the broader category of ensemble visualization techniques (see Wang *et al.* [WHLS18] for a recent overview). Notably, while ensemble visualization has emerged as an important field, to the best of our knowledge, visualization techniques for ensembles of 2D

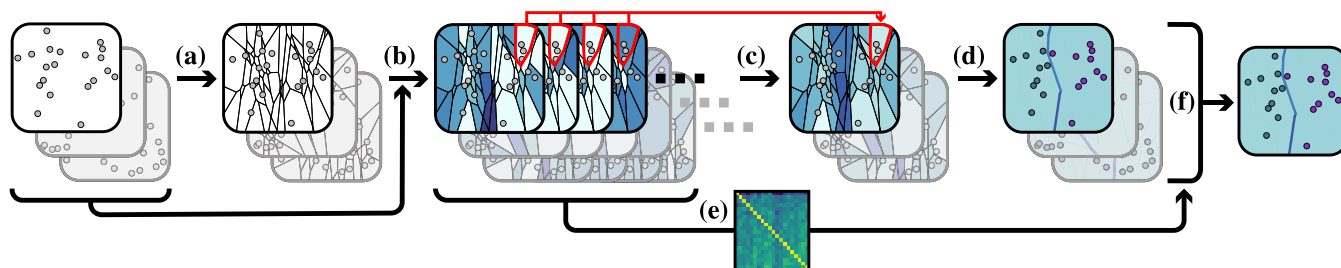


Figure 2: Method overview. (a) A k -order Voronoi diagram is computed for each ensemble member. (b) Each pairing of a projection P with a Voronoi diagram \mathcal{V} yields a coloured version of \mathcal{V} where the colour of a cell encodes the stability of its corresponding k -neighbourhood in P . (c) Coloured variants of \mathcal{V} are aggregated into a single colouring. Colours encode the stability of each cell with respect to the ensemble. (d) Core features of each Voronoi diagram are extracted via density-based clustering—identifying local maxima of stability—and used to generate robustness plots. (e) A pairwise similarity matrix based on k -neighbourhoods is computed from stability values for each pair (P, \mathcal{V}) which then allows us to identify one or more clusters and select representative robustness plots in (f).

points are not existing. The majority of works in ensemble visualization have addressed ensembles of physical fields, or features derived from such fields, and they have focused on the extraction and visual encoding of their variability. Parametric statistical distributions and distribution shape descriptors for scalar-valued ensembles were presented by Love *et al.* [LPK05]. Isocontours in an ensemble of 2D scalar fields have been conveyed via spaghetti plots [PWB*09, SZD*10, Wil11]. Different variants of confidence regions were introduced to represent the major geometric trends in ensembles of isocontours and streamlines [WMK13, MWK14, FBW16, FKRW16]. Demir *et al.* [DJW16] proposed a closest-point representation to convey the central tendency of an ensemble of multi-dimensional shapes. Hummel *et al.* [HOGJ13] analyse the spread of particle trajectories in an ensemble of vector fields to reveal the transport variability, and Jarema *et al.* [JDKW15] model directional data ensembles via mixtures of probability density functions to compactly classify complex distributions. Poethkow and Hege [PH13] and Athawale *et al.* [ASE16] use location-wise estimators of non-parametric distributions from ensemble members and use these distributions to estimate the spread of surface and vector field features.

Alternatively, clustering has been used to group ensemble members regarding similar data characteristics [BM10, TN14, OLK*14, FBW16, FFST19]. While these techniques compare ensemble members to each other, our approach aims at finding groups of elements in each member which remain ‘close’ to each other in all members. Strehl and Ghosh [SG02] apply different clustering techniques to one single ensemble, and combine the results into a single clustering. Ferstl *et al.* [FKRW16] cluster different time steps of the same ensemble in a hierarchical way to convey the change of clusters over time. For the clustering of genomic data, Lex *et al.* [LSP*10] introduce extended parallel coordinate plots to compare different clusterings and analyse the quality of cluster assignments. Kumpf *et al.* [KTB*18] consider a set of ensembles and cluster each ensemble using the same algorithm and number of clusters. Clusters are matched to clusters in a reference clustering by using their intersection, and cluster variability plots are proposed to analyse the robustness of the clustering results. Our method, in contrast, does not make any assumption about whether the data can be clustered and into how many clusters it can be separated.

In our scenario, each ensemble member is a set of 2D points, which is generated by applying a dimensionality reduction technique to a set of points in a high-dimensional space. Some recent surveys [KH13, LMW*16] give thorough overviews of visualization techniques for high-dimensional data. In combination with dimensionality reduction, clustering is often used to identify groups of points lying close together in the low-dimensional space or forming coherent structures in this space. Wenskovitch *et al.* [WCR*17] discuss the combination of dimensionality reduction and clustering techniques, and provide recommendations for their concurrent use. General overviews of the numerous techniques for dimensionality reduction and clustering can be found in the surveys by Sorzano *et al.* [SVM14] and Everitt *et al.* [ELLS11], respectively.

Related to our approach are also techniques which aim to find projections that best represent the structures in high-dimensional data, by using quality measures for projections [FT74, HA85]. Even though the goal of these techniques is different to ours, as we do not attempt to find the best projection for a given dataset, proposed measures indicate the (dis-)similarity between projections and might be used for robustness analysis as well. Examples include vector distance measures for high-dimensional feature descriptors [BvLBS11] and feature vectors derived from point-wise distance matrices [JHB*17], as well as measures using matrix norms to quantify the dissimilarity of multivariate projections invariant to affine transformations [LT16]. None of these approaches, however, analyses the stability of multiple projections. Instead, they quantify the similarity of projections to order them or find new projections with as less as possible redundancy to previous ones.

3. Method Overview

Let X be a finite set of n entities in some multi-dimensional coordinate or parameter space. Our method analyses an ensemble \mathcal{E} of embeddings of X into the 2D plane. Each embedding is an ordered 2D point set $P = (p_x)_{x \in X} \in \mathcal{E}$ of cardinality n . Points in different ensemble members that have the same index refer to the same entity.

To analyse whether certain subgroups of points are stably connected—using a suitable neighbourhood relation—across all ensemble members, we introduce the pipeline in Figure 2. In general,

for a certain subgroup of size k , all $\binom{n}{k}$ available subgroups need to be tested. As this is intractable, we utilize k -order Voronoi diagrams—computed in step (a)—to reduce this number to $\mathcal{O}(kn)$ many k -neighbourhoods. The k -order Voronoi diagrams provide topological information that is used to assemble neighbourhoods of fixed size k into subgroups of arbitrary size. The geometric structure of these diagrams is further used to visualize simultaneously the stability information that is obtained from it and a 2D embedding of X .

In step (b), the stability of each k -neighbourhood is assessed using minimum covering circles of the corresponding point set in all other ensemble members. In this way, multiple stability values are obtained for each neighbourhood. These values are finally aggregated into a single value that quantifies the overall stability of a neighbourhood in the ensemble (step (c)).

Higher order Voronoi diagrams can be used in principle to perform a fine-granular analysis of the neighbourhood relations in the ensemble. However, since they comprise thousands of convex polygons which cannot easily put into relation to the data points, some expertise is required to observe the major trends. To ease the interpretation, an abstract visualization is provided in which the information is condensed. Therefore, the major information regarding stable subgroups and their relations to each other is extracted by clustering the k -order Voronoi diagrams (step (d)). The clustering is only used to obtain an alternative visual representation, and it incorporates derived stability information of subgroups. A region's stability is mapped to its background colour, and ridges between these regions are coloured to indicate how strongly adjacent subgroups are separated. We will subsequently call these plots *robustness plots*. A robustness plot allows to identify points which are connected via k -neighbourhoods of locally high stability. Furthermore, if regions of high stability have a particular shape, such as band- or ball-like, one can conclude that these shapes also appear in most of the ensemble members.

For each $P \in \mathcal{E}$, a distinct k -order Voronoi and robustness plot is generated. As demonstrated in Section 5.7, many of these plots carry similar information, in particular regarding the stable regions. Building upon a pairwise similarity matrix that is computed in step (e), a small set of representative plots is finally selected in step (f). These plots carry most of the stability information, and they are used to embed the stability information into the 2D domain. A robustness plot of any ensemble member reveals the agreement of the member to the derived stability information.

4. Stability Analysis

Our notion of stability is based on how well neighbourhoods are preserved in different point sets: Given a k -subset V of points in one point set, how large is the spread of these points in all other point sets. This is determined by computing the smallest possible disks that cover the points in V in the other sets. These disks may also cover points not being part of V , indicating that additional points interfere with the neighbourhood relationship. The stability value $\text{stab}(V, P)$ of V with respect to $P \in \mathcal{E}$ is then computed as the ratio between k and the number of points covered by the disk. Consider P containing the grey and orange points in Figure 3(a), and V

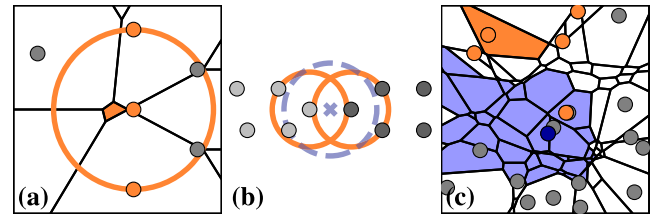


Figure 3: (a) A three-order Voronoi diagram of six points. One cell, its nearest three neighbours (orange points) and the smallest disk enclosing these points are shown. (b) Orange circles indicate the 3-neighbourhoods of their centre points. The 3-neighbourhood centred at the location marked with the blue cross is indicated by the dashed blue circle. (c) Five-order Voronoi diagram of a random set of 20 points. Orange points are the five nearest points to the orange cell. All cells related to the dark blue point are coloured blue.

containing the three orange points. Then, $\text{stab}(V, P)$ evaluates to $3/5$, since the cardinality of V is 3 and the smallest disk covering V contains five points. This measure is invariant to uniform scaling and isometric transformations applied to P , and it has domain $(0, 1]$, as the disk always contains at least the k points of V . We extend this measure to an ensemble of point sets $\mathcal{E} = \{P_1, P_2, \dots, P_\ell\}$, by using the mean over all $\text{stab}(V, P_i), i \in [1, 2, \dots, \ell]$. Alternatively, aggregations that better represent the distribution of stability values can be used.

4.1. Voronoi diagrams

The subsets V that are considered in the stability analysis are computed from the higher order Voronoi diagrams. To reduce the number of neighbourhoods to be computed, the analysis could, in principle, be restricted to the k -neighbourhood around each initial point, i.e. the k -neighbourhood including the point and its $(k - 1)$ -nearest neighbours. However, as shown in Figure 3(b), this approach does not always allow for a meaningful assessment of the stability of subgroups in the point set. In the example, when grouping each point with its two nearest neighbours, no 3-subset covering at least one light grey and one dark grey point is obtained. In particular, stability values computed for these subsets are not affected by the distance between the light and dark grey group in other point sets of the ensemble \mathcal{E} . Thus, information about the stability of the relative position of the dark grey group to the light grey group cannot be inferred. Both may be adjacent only in the seen point set, or in all point sets of \mathcal{E} .

Ideally, the situation in Figure 3(b) can be resolved by sampling the 3-nearest neighbours of the location marked by a blue cross, so that points with different colours are related to each other. In the general case, this approach leads to sampling all k -nearest neighbourhoods induced by every possible location in 2D space. When grouping all 2D locations according to their k -nearest neighbourhoods, a partition of the 2D space into a finite number of (possibly unbounded) convex regions is obtained. These regions are called Voronoi cells. A many-to-many correspondence between cells and points is established by relating a cell to each of its k -nearest points (see Figure 3c). The partition is called the k -order Voronoi

diagram of the given point set and extends the well-known (one-order) Voronoi diagram.

Considering region-wise neighbourhoods provides several benefits over the approach of only sampling neighbourhoods centred around 2D points. Firstly, it does convey strictly more information by construction since all 2D points may also be considered 2D locations. Secondly, it defines a meaningful spatial relationship by relating two k -neighbourhoods iff their corresponding Voronoi cells share a common facet. Adjacent neighbourhoods of 100% stability are guaranteed to be located next to each other in every point set. Finally, the computation of k -order Voronoi diagrams is tractable for modest values of k , with run-time complexity $\mathcal{O}(k^2 n \log n)$, and yields $\mathcal{O}(kn)$ neighbourhoods [Lee82].

In our approach, k -order Voronoi diagrams are computed in step (a) via the algorithm by Lee [Lee82]. It gradually generates Voronoi diagrams of increasing order by cutting cells of the previous order $k - 1$. As the $k - 1$ nearest neighbours for a cell are already known, the remaining k -th neighbour is computed by intersecting the cell with the one-order Voronoi diagram obtained when discarding the $k - 1$ known neighbours. Then, the resulting cut-outs with similar k -neighbourhoods are stitched together. It remains to compute one-order Voronoi diagrams for which we use the *Qhull* implementation [BDH96].

When computing smallest disks for k -neighbourhoods in a Voronoi diagram, the iterative approach of Lee greatly increases the performance. Each cut-out originates from a cell of order $k - 1$, for which the smallest encompassing disk D of its neighbourhood regarding a point set P has already been computed. If the added k -th point, embedded in P , is also contained in D , the disk D_{new} for all k points is again D . If not, the point has to be located on the boundary of D_{new} . Since each circle is entirely defined by three points, D_{new} can be computed in $\mathcal{O}(1)$ whenever three or more cut-outs are stitched together. When a k -order cell is generated by stitching two cut-outs—less is not possible—a possibly existing third boundary point has to be guessed in $\mathcal{O}(k)$. In practice, approximately every second cell is stitched by three or more pieces.

4.2. Clustering stable subgroups of points

As can be seen in Figure 3(c), the interpretation of higher order Voronoi diagrams is challenging. First of all, the size of a Voronoi cell, in general, does not provide relevant information. In addition, since related cells and points may be distant from each other, especially for high orders or areas with low point density, it becomes difficult to infer these relations. Finally, the fine granular cell structures introduce visual clutter and even interfere with the basic stability information to be conveyed (see Figure 1(a2)).

To ease the interpretation of k -order Voronoi diagrams, density-aware clustering is used (step (d) in Figure 2). The approach builds upon the observation that in k -order Voronoi diagrams, when their cells are coloured according to the stability of neighbourhoods, regions packed with stable, connected components are often separated by instable bands. This property is seen in Figures 1(a), 2 and 13.

By means of clustering, these components are extracted and translated to subgroups of stable points. Hence, considerably larger stable

areas than k -neighbourhoods can be identified and put into relation to each other. Two points are located in the same cluster if they are part of the same stable submanifold, whereas a submanifold is considered stable if it occurs in most of the point sets of the ensemble. The topological structure of the submanifold is outlined by ridges.

4.2.1. Clustering of Voronoi cells

From a k -order Voronoi diagram, enriched by stability information, connected areas of locally high stability can be emphasized by arranging the Voronoi cells in a hierarchical manner. For this, we utilized concepts from persistent homology [EH08, EH10]. More specifically, the function mapping Voronoi cells to their respective stability values can be interpreted as a monotonic function defined on a simplicial complex. Its merge tree yields the desired hierarchical representation, and a persistence value attached to each node in the tree is used to select the final clustering.

Let $\lambda \in [0, 1]$ be an arbitrary threshold. For a selected λ , the *superlevel set* $L^+(\lambda)$ is the set of all Voronoi cells with a stability value greater or equal to λ . The superlevel set is composed of a set of connected components $\text{Comp}(\lambda)$. When λ is decreased towards 0, these components merge until one connected component remains. By relating a component to all components that have been merged to form it, a tree with nodes $\bigcup_{\lambda \in [0, 1]} \text{Comp}(\lambda)$ is generated that serves the desired hierarchical representation. For instance, the Voronoi diagram in Figure 5 yields the hierarchical representation that is shown by the Dendrogram (1) in Figure 4.

In principle, and similar to some common clustering algorithms like Agglomerative Hierarchical clustering and Density-based Spatial Clustering of Applications with Noise (DBSCAN), a clustering can be obtained from a certain level of the merge tree. If clusters are extracted from the same level, however, prominent clusters at other levels will be missed. This situation commonly occurs when one cluster is located in a vastly stable region of the Voronoi diagram (cluster ● in Figure 1c), and a rather unstable cluster is surrounded by significantly less stable cells (cluster ● in Figure 1c).

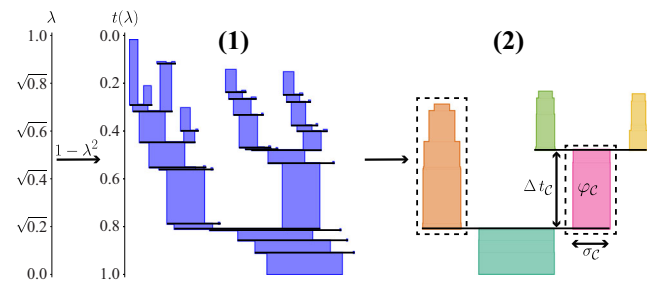


Figure 4: The cluster hierarchy for the Voronoi diagram in Figure 5. Each box indicates a connected component with the width encoding its significance. Horizontal lines signal merge events. By distorting the stability threshold scale so that the timing scale becomes linear, the height of a box encodes the lifetime of a component and its area corresponds to persistence. Dendrogram (2) displays the simplified hierarchy. The framed clusters form the largest disjoint component set regarding overall persistence. Box colours match the components in Figure 5(a).

To avoid this limitation, we utilize the same concept as used in the clustering technique Hierarchical Density-based Spatial Clustering of Applications with Noise (HDBSCAN) [CMS13]. Instead of taking clusters from the same level, HDBSCAN slices the merge tree at different depths such that components which persist for an extended duration before merging into another one are kept intact. To detect persistent components, sufficiently small components and their merge events are dropped beforehand.

When HDBSCAN is used in the current scenario, the size of a component can be determined by the number of encompassed Voronoi cells. However, since we ultimately aim for clustering points but not Voronoi cells, we introduce a different notion of size, called *significance*. The significance of a Voronoi cell roughly represents by that cell (see Section 4.2.2). Then, the significance of a connected component is the sum of significances of its encompassed Voronoi cells. The box width in Dendrogram (1) of Figure 4 encodes the significance of each component.

The modified HDBSCAN algorithm now works as follows: Firstly, the hierarchical representation is simplified and made robust against noise by dropping connected components. In our case, it is reasonable to drop all components with a significance lower than the order k of neighbourhoods. This process results in Dendrogram (2) in Figure 4. In a second step, a persistence term φ_C is assigned to each connected component based on its significance integrated over its lifetime Δt_C . In Figure 4, the area of each component represents its persistence. The time $t(\lambda)$ of a merge event is derived from the value of λ when the event occurred. The negative of the derivative $-\partial t(\lambda)/\partial \lambda$ serves as an unnormalized weight distribution along the domain of λ , and the lifetime between $t(\lambda_2)$ and $t(\lambda_1)$ reduces to integrating weights along $[\lambda_1, \lambda_2]$ by

$$t(\lambda_2) - t(\lambda_1) = \int_{\lambda_1}^{\lambda_2} -\frac{\partial t(\lambda)}{\partial \lambda} d\lambda.$$

We suggest to use $t: \lambda \mapsto 1 - \lambda^2$ with the weight distribution $-\partial t(\lambda)/\partial \lambda \propto \lambda$ to favour stable regions. In all our experiments, this lead to the most plausible clustering results. Depending on the application, the use of other weight distributions such as a constant distribution—yielding $t: \lambda \mapsto -\lambda$ —may be worth considering as well.

Finally, a set of disjoint connected components is selected such that the sum of their persistence values becomes maximal. The selected set represents the final clustering of the Voronoi cells, of which—as in DBSCAN—some may not be covered and thus considered noise. In Figure 4, the selected components are framed by dashes. As seen in Figure 5(a), the components do not cover the unstable cells, thus letting the user instantly distinguish between stable and unstable groups of points when they are overlaid.

4.2.2. Point clusters

To relate the Voronoi cell clusters to the initial point set, these clusters are interpreted as fuzzy point clusters, and hard point clusters are derived by a fix assignment of points to their most likely cluster. Since an initial point can be part of many equally stable neighbourhoods, it contributes fractions to each cell C the point is related to.

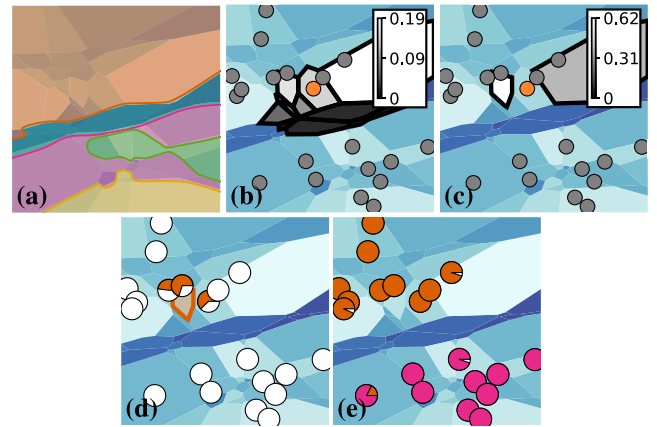


Figure 5: Extracting point clusters from a three-order Voronoi diagram. (a) Connected components of Voronoi cells in the simplified cluster hierarchy shown in Figure 4. (b) The contributions of the orange point are indicated by overplotting the related cells with their respective contribution values. (c) Contributions after concentration. (d, e) Pie charts illustrate proportions of a point's contribution to (d) the highlighted cell (orange) and all other cells (white), and (e) fuzzy point clusters stemming from the framed Voronoi cell clusters of Figure 4. White slices indicate contributions to cells considered noise.

Fractions are chosen such that they are proportional to $\text{stab}(C, \mathcal{E})$ and add up to one. The contributions of an exemplary point are visualized in Figure 5(b).

Compared to assigning the whole contribution of a point to a single cell, however, this approach smooths out densities over adjacent cells. As a consequence, it prevents HDBSCAN from identifying meaningful clusters. To avoid this, we decided to let the contributions of a point p ‘flow’ upwards to adjacent, related cells with higher stability until the whole contribution is concentrated in cells with spatial-local maximal stability. Applying this process to the situation in Figure 5(b) gives the concentrated contributions shown in Figure 5(c).

A Voronoi cell can now be related to a fuzzy point cluster by gathering the contributions from points contained in its k -nearest neighbourhood. For a single cell, the sum of all contributions defines its significance (see Figure 5d). Note that the contribution of the highlighted point in Figure 5(c) to the marked cell in Figure 5(d) is visualized in both figures. A Voronoi cell cluster may now be interpreted as a fuzzy point cluster, by adding up all contributions of the cells contained in the cluster. When applying the process to the selected Voronoi cell clusters of Figure 4, the fuzzy point clustering depicted in Figure 5(e) is obtained. Defuzzifying yields the expected clustering that separates points contributing to the upper and lower Voronoi cells, respectively.

4.3. Visualizing stability in point sets

We propose an abstract visualization of k -order Voronoi diagrams which refrains from the limitations of k -order Voronoi diagrams mentioned in Section 4.2. In addition to showing the point set in a

scatterplot, points in different clusters as well as points deemed noise are separated via ridges. The ridges are extracted by querying the (regular) one-order Voronoi diagram. Here, each ridge is a boundary between two adjacent cells, each containing a single point. If the points fall into different clusters, the ridge is drawn. The ridge colour is determined by the least stable cell that is touched when following the line connecting both points in the original k -Voronoi diagram. It encodes how strongly both clusters are separated. The background colour of a cluster is defined by a weighted average of the stability values of the cells assigned to the cluster, computed by

$$\frac{\sum_{\mathcal{C}} \omega_{\mathcal{C}} \text{stab}(\mathcal{C}, \mathcal{E})}{\sum_{\mathcal{C}} \omega_{\mathcal{C}}} \quad \text{with } \omega_{\mathcal{C}} := \sigma_{\mathcal{C}} \cdot \Delta t_{\mathcal{C}},$$

where $\sigma_{\mathcal{C}}$ is the significance of cell \mathcal{C} and $\Delta t_{\mathcal{C}}$ its lifetime in the cluster. Thus, cells contributing most and for the longest time to the cluster are emphasized.

The colourmap of choice originates from a subrange of the perceptually uniform cmocean ice colourmap [TGH*16]. It emits a pleasing light colour for highly robust regions while effectively revealing small changes in robustness. Both factors are especially important if most of the screen is filled with areas depicting high robustness. The background of noise clusters is coloured in grey. It can be clearly distinguishable from the chosen colourmap without introducing any undesirable signaling effect. Points are coloured according to cluster membership. Assigned colours have been carefully selected to be perceptually discriminative without conflicting with the background.

4.4. Selection of representative point sets

Since a single plot for each ensemble member would overburden the visualization, we guide the user's selection process of viewing certain point sets in step (f) by computing a pairwise similarity matrix in step (e) of the pipeline. The pairwise similarity between point sets $P, Q \in \mathcal{E}$ encodes how well the k -neighbourhoods arising in Q are preserved in the point set P . It is computed by using the stability measure $\text{stab}(V, P)$ proposed in Section 4. The stability values of all neighbourhoods V occurring in the k -order Voronoi diagram of Q are averaged.

Since the k -order Voronoi diagrams of P and Q contain different neighbourhoods, interchanging the roles of P and Q changes the result. This makes the measure non-symmetric. It is also not reflexive in situations as in Figure 3(a), where the centre of the smallest disk encompassing a k -neighbourhood V is not located in the k -order Voronoi cell associated to N . When comparing the point set with itself, its three-order Voronoi diagram contains the orange coloured cell. It relates to the neighbourhood formed by the orange points with its stability given by $3/5$ (see Section 4). Thus, the average of stability values over all cells is strictly lower than one. However, in all of our experiments, strong symmetry and a sharp diagonal in the similarity matrices are observed (see, for instance, Figure 6).

To extract clusters of similar point sets in the ensemble \mathcal{E} , matrix seriation [BBHR*16] is used to sort columns and rows such that hidden patterns become apparent. We utilize the two rank ellipse seriation of Chen [Che02] for identifying blocks of point sets, as

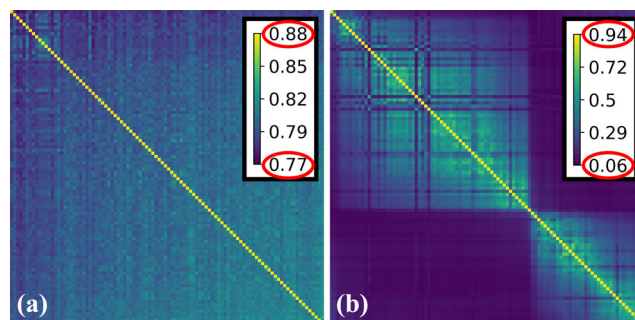


Figure 6: Non-symmetrized similarity matrices for the 100 projections (a) of the Cloud dataset, ordered according to their PageRank scores, and (b) of the MNIST dataset, ordered by the rank-2 ellipse seriation technique of Chen [Che02]. For MNIST, our measure suggests to form two groups of similar projections.

shown in Figure 6(b). When the blocks are identified, for each block, a representative is picked that is determined by the highest score when applying the PageRank algorithm [BP98].

5. Results and Evaluation

We now analyse the stability of k -neighbourhoods in ensembles of 2D point sets that are generated via dimensionality reduction techniques from high-dimensional data points. To generate the robustness plots, DBSCAN and k -means are used for clustering. Performance measurements are performed on a standard desktop PC, equipped with an Xeon CPU E5-1650 v2 with 6 cores @ 3.50GHz, 32 GB RAM, and an NVIDIA GeForce GTX 1070 Ti graphics card with 8 GB VRAM.

The following datasets are used:

- **MNIST:** A subset of the MNIST dataset [LBBH98]. It is composed of 1k 196-dimensional points, each representing the 196 activations in the second layer of a convolutional network that was trained to classify input images to digits 0–9. Since all digits occur equally often in the 1k inputs, 10 equally large clusters can be expected.
- **Cloud:** A 3D multi-parameter cloud simulation [WBJ*18]. The dataset contains roughly 11k points, each representing a 12-dimensional vector showing physical parameters such as ice, graupel and water contents at different locations in the 3D simulation domain. It is unknown whether the dataset contains clusters.
- **Gaussian:** A synthetic dataset, generated by sampling a mixture of eight isolated multi-variate Gaussian distributions in 3D space with random covariance matrices. The dataset contains 1k points equally spread over all distributions.

For each dataset, an ensemble of 2D points is created via MDS or Barnes-Hut t-SNE [VDM14] with randomly generated initial point positions. One hundred different random initializations are used to represent the ensemble variability. For a certain dataset, this number can be adapted by generating new random initializations until the average of the pairwise distances introduced in Section 4.4 falls below a threshold. The size k of the neighbourhoods we aim

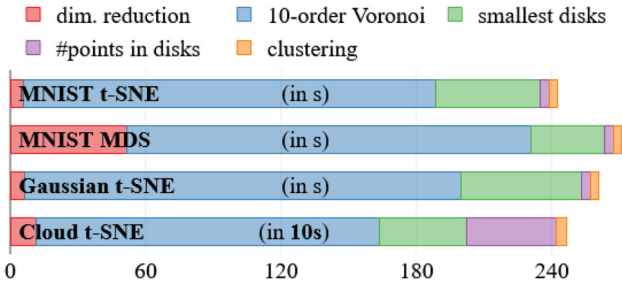


Figure 7: Performance measurements of our pipeline for computing a robustness plot in a single thread.

to analyse is set to 10, independently of the dataset. Due to space restrictions, we analyse only the overall best ranked projection of each dataset and do not consider other representations of, e.g. the MNIST dataset projected with MDS.

The times required to generate the Voronoi plots using a single thread on our target architecture are given in Figure 7. The approach scales approximately linearly, only degraded by the quadratic runtime behaviour of naively computing the smallest disk enclosing for a certain neighbourhood. By employing a hierarchical acceleration structure, the complexity of neighbour identification can be decreased to $\mathcal{O}(n \log n)$. For computing the pairwise similarity matrix, all stages listed in Figure 7, except for the clustering stage, are required. They can be executed for all projections in parallel.

5.1. Choice of k

The parameter k controls the locality of our proposed method. By construction, our approach cannot detect stable subsets of maximal size $< k$ since each k -neighbourhood containing a stable subset of smaller size also contains some unstable points enforcing low stability values. On the other hand, as we assemble stable regions by collecting circular shaped neighbourhoods, all detected regions have to be covered by disks not containing instable points. If the parameter k , and thus the disks, become too large, stable regions of intricate shape may not be detected anymore. For this reason, we favour a small value for k in our experiments.

To detect a stable group of points, our approach requires this group to be locally stable as well, i.e. its k -neighbourhoods must be stable over the entire ensemble. For instance, by increasing k from 10 to 60 in the example shown in Figure 8(a), the points are bisected into two stable regions (light background). For $k = 10$,

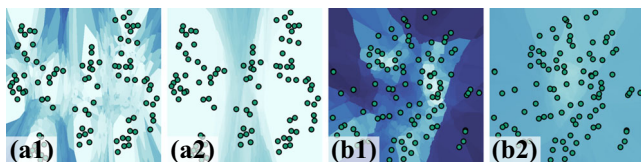


Figure 8: Voronoi plots of two small-scale datasets (100 points) with (x1) $k = 10$ and (x2) $k = 60$.

this information is lost. Instead, the background reveals the stability of smaller subgroups. If no large stable subgroup exists, such as in Figure 8(b), increasing k smooths out the patterns arising for smaller k . Animations with k ranging from 2 to 100 are provided in the Supporting Information.

We suggest to inspect several point sets of the ensemble \mathcal{E} to determine a reasonable value of k . By covering prominent structures with sufficiently small disks, one can compute the average amount of points covered per disk and choose this value as an initial guess for k . Due to the iterative nature of our approach, k may be lowered without costly re-computations. In practice, the magnitude of k is limited by the quadratic complexity in k of computing Voronoi diagrams (see Section 4.1).

5.2. Cluster variability plots

In recent work by Kumpf et al. [KTB*18], a method for analysing the variation in composition of clusters over multiple clusterings and the variability in cluster membership for individual ensemble members was introduced. The method takes a set of ensembles, each composed of a number of high-dimensional data vectors, and then clusters these data points separately for each ensemble. The method assumes that the number of clusters is known, and a reference clustering exists to which all clusters can be matched. Via pie glyphs, the method encodes for each data point the frequency of cluster membership over the set of clusterings (see Figures 10a,b and 11b). Stable clusters are indicated by groups of glyphs coloured predominantly with the same colour, with no or only a small number of pieces with different colours. Highly fragmented pie glyphs indicate frequent changes in cluster assignments over the ensemble, hinting to instable groups. By picking a glyph, all data points with similar cluster membership over all clusterings can be highlighted.

Since the method strongly relies on the clustering results, including the selected number of clusters and clustering algorithm, it can obscure the relationships between data points. This is demonstrated in Figure 9, where each member of an ensemble of three 1D point sets including eight elements is clustered via k -means. Even though the ensemble members are only slightly jittered without changing the neighbourhood relationships, the clustering algorithm—depending on the point positions as well as a random initial positions of centroids—groups different sub-sets together. Hence, the visualization using pie charts (bottom row in Figure 9) exhibits a transitioning artefact. It suggests that both clusters are connected in most ensemble members, and thus, form a single, connected cluster in the ensemble. In the following experiments, we demonstrate the

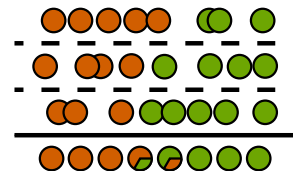


Figure 9: Top rows: Three 1D point sets, each split by k -means into two clusters. Bottom row: the resulting pie glyphs of the method of Kumpf et al. [KTB*18].

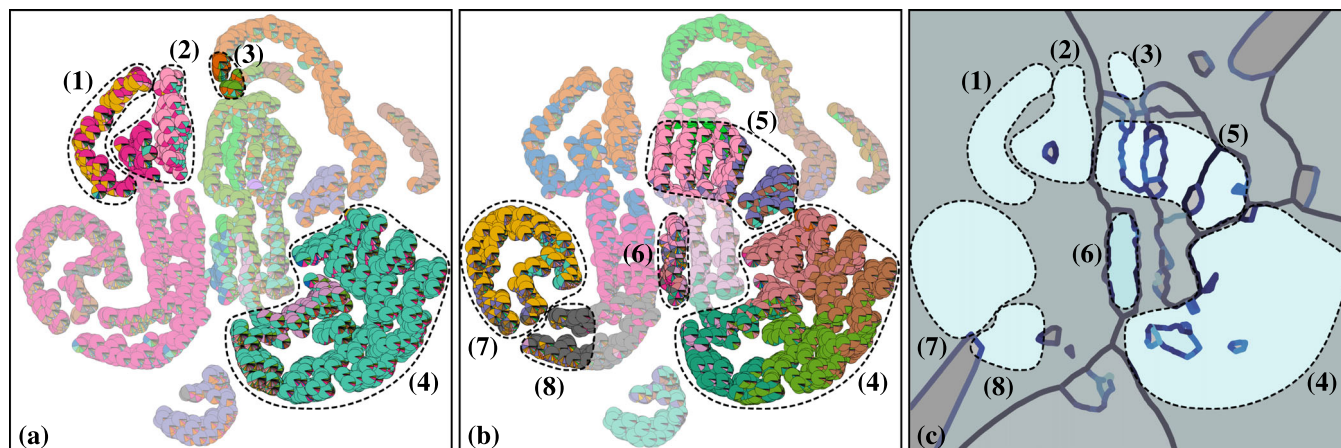


Figure 10: (a, b) Member-centric cluster variability plots [KTB*18] and (c) the robustness plot (Figure 1(b) without points) of the Cloud dataset projected with t-SNE. Clusters are formed by either (a) DBSCAN or (b) k-means. Regions of special interest are highlighted.

different strengths and weaknesses of the method by Kumpf *et al.* in comparison to Voronoi plots. We will also show that both visualizations can be combined effectively, the Voronoi plot makes use of the visual background channel not occupied by Kumpf *et al.*

5.3. Cloud dataset

The robustness of t-SNE to variations in its input configurations is analysed in the following. For the Cloud dataset, the pairwise similarity matrix renders all projections equally similar (see Figure 6a). Thus, a single representative projection is selected via the PageRank algorithm (see Section 4.4). Corresponding Voronoi and robustness plots are displayed in Figures 1(a) and (b). The visualizations show that t-SNE is largely robust, and only a few scattered outliers are produced. As seen in the region dominated by ● points in Figure 1, some clusters can be deemed adjacent in many projections since no ridges form. Conversely, a cluster enclosed by dark ridges (● in Figure 1) is not considered adjacent to its currently depicted neighbouring clusters in many projections.

To compare Voronoi plots to the pie glyphs by Kumpf *et al.*, we use their method in combination with the DBSCAN clustering algorithm. The use of DBSCAN is motivated by the band structures in the projections, from which the existence of manifold structures in the original data can be concluded. Although the clusterings of individual projections look meaningful, it is difficult to relate them to each other over multiple projections. Due to different linkage thresholds in each projection, vastly different split and merge events occur, and the number of computed clusters varies. This makes it impossible to compute meaningful matchings between different clusterings. As a consequence, many pie glyphs are highly fragmented (in Figure 10a). Over that, transitioning artefacts, as seen in region (1) and discussed in Section 5.2, are introduced and need to be resolved.

Voronoi plots, on the other hand, do not suffer from these deficiencies, as they do not rely on separate clusterings for each projection. In Figure 10(c), areas of high stability are reliably determined. Two dominant cluster colours are observed in region (2) of Figure 10(a), yet the robustness plot (Figure 10c) does not show a ridge between them. When analysing projections manually, one

sees that both clusters indeed are located next to each other in all projections, confirming the lack of a ridge in the robustness plot. In comparison, pie glyphs show multiple dominant colours, but not a smooth transition between them. Hence, spatial proximity cannot be concluded solely from the pie glyphs. DBSCAN fails to reliably extract this information, since both groups of points are often separated by a small area of zero density. The same mismatch holds true for region (3).

The result when using k-means with 16 clusters instead of DBSCAN is shown in Figure 10(b). Region (4) now splits into four clusters, which can only be joined manually by following smooth transitions. Transitions, on the other hand, can be misleading. For instance, one could assume a transition between colours ● and ● in region (5). When analysing the projections in combination with the robustness plot, it turns out that points being mainly dark blue move independently of the rose ones. Furthermore, region (6) is perceived as highly unstable. Yet, besides being broken apart into two or three components from time to time, the components themselves are isolated and stable when browsing the ensemble of projections. Thus,

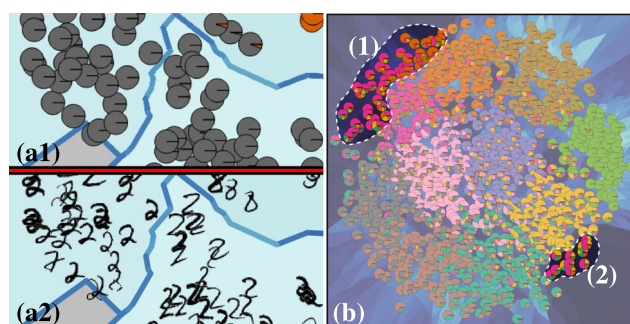


Figure 11: (a) Excerpt of the best ranked MNIST t-SNE-projection underlaid with its robustness plot. Points are rendered as (a1) pie glyphs or (a2) their corresponding MNIST digit representation. (b) Member-centric cluster variability plot [KTB*18] of the best ranked MNIST MDS-projection together with the Voronoi background of Figure 1(c). Regions of special interest are highlighted.

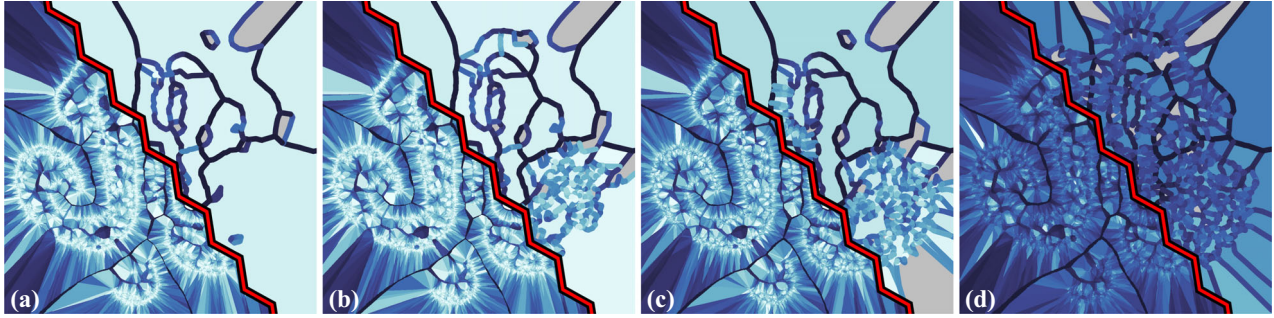


Figure 12: Resulting Voronoi and robustness plots for the best ranked Cloud projection with each point perturbed by up to (a) 0.15, (b) 0.2, (c) 0.5 and (d) 2.0 in each direction.

the robustness plot correctly classifies this region as a slightly less stable and strongly isolated cluster. The robustness plot also captures the proximity of the crescents in region (7), and it reveals that they are always placed next to points of region (8) with only a small strip of empty space between them. Pie glyphs fail to communicate this relation.

5.4. MNIST dataset

The assumption of 10 clusters in the MNIST dataset—one for each digit 0 to 9—suggests using the method by Kumpf *et al.* [KTB*18] to analyse the stability of these clusters. In our implementation, we use k-means for clustering the initial 2D point sets.

In particular, it can be seen that the method of Kumpf *et al.* fails to reveal certain anomalies, i.e. small inter-cluster details. The pie glyphs in Figure 11(a1) suggest a strongly stable cluster. The robustness plot in the background, on the other hand, shows a separating ridge of medium stability. When plotting the input digits corresponding to each data point (see Figure 11a2), we observe that the robustness plot separates the hand-written digit 2 with and without loops. By following the cluster over the ensemble, we discover that the cluster of digit 2, in fact, consists of two adjacent sub-clusters warping independently of each other.

From the structure and colouring of the pie glyphs in Figure 11(b), stable and instable regions, outliers as well as transitions between clusters can be concluded. For the MNIST dataset, the Voronoi plot in the background conveys mostly similar information (Figure 1c shows the Voronoi plot and derived groups of stable points). However, Voronoi plots, in this case, are not effective in relating distant points to each other. Although we point out instable neighbourhoods in regions (1) and (2), only due to the pie glyphs, one can detect that both regions regularly exchange some points.

5.5. Gaussian dataset

For the Gaussian dataset, the plot in Figure 13 reveals all isolated Gaussians. In particular, the following well-known properties of t-SNE can be observed. Firstly, t-SNE tries to preserve neighbourhoods so that points of the same isolated Gaussian are always clustered together. Highly stable regions around each Gaussian confirm this assumption. Secondly, the arrangement of the eight clusters is

random since no neighbourhood considered by t-SNE touches two Gaussians simultaneously. The plot reveals this instability by dark ridges separating the Gaussians.

5.6. Sensitivity of k -order Voronoi diagrams

To investigate the sensitivity of Voronoi plots to small perturbations in the reference point set, the following experiments are performed: Each point is jittered from its 2D location by an offset uniformly sampled from $[-x, x]^2$. x is increased continually from one experiment to the next. We choose $x \in \{0.01, 0.02, 0.05, 0.1, 0.12, 0.15, 0.2, 0.5, 1.0, 2.0\}$, with 0.12 being the average nearest neighbour distance of all selected projections. Only the transitions with the most significant changes are shown in Figure 12. The unperturbed plots are shown in Figures 1(a) and (b).

Up to a box radius of $x = 0.2$, one can hardly perceive any difference in the coloured k -order Voronoi diagrams, although already for $x = 0.1$, over 50% of all Voronoi cells in the unperturbed diagram are lost. For values $x > 0.2$, the stability values start to degrade notably. Note that even for $x = 2.0$, the Voronoi plot still shows the same ridge structures. When inspecting the robustness plots, significant changes cannot be observed up to $x = 0.15$. Starting at $x = 0.2$, the clustering approach starts to fragment the previously generated clusters. We attribute this behaviour to the use of a weight

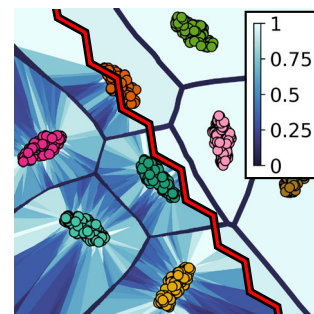


Figure 13: Split showing the Voronoi and robustness plots for the best Page-ranked t-SNE-projection of the Gaussian dataset. The point colours encode the eight isolated Gaussians in the dataset.



Figure 14: Voronoi plots with point colours according to the clusters found in the best Page-ranked projection of the according Dataset. (a) The best ranked and (b) a randomly picked Cloud projection. (c) Random MNIST MDS-projections of the larger group of similar projections in Figures 6(b) and (d) smaller group, respectively.

distribution favouring high stability in Section 4.2.1. While numerous small, highly persistent components are maintained, the overall persistence of larger components degrades. As soon as the persistence of a larger component drops below the accumulated persistence of its subcomponents, it fractures. This effect can be counteracted in principle, by either exchanging the weight distribution or by raising the significance threshold when simplifying the component hierarchy.

Since we do not think that inspecting globally unstable projections is particularly meaningful, we adhere to the suggested weight distribution and keep the significance threshold as low as reasonable. Overall, we conclude that the proposed method is robust to noise of a magnitude similar to the average nearest neighbour distance. Thus, adjacent points may flip positions without significant changes.

5.7. Verifying representative projections

We further evaluate how well the selected representative projection emphasizes the major trends in the data. Therefore, different projections are compared by mapping point colours in a random projection to the clustering of similar k -neighbourhoods obtained from the best ranked projection. The results are shown in Figure 14.

Notably, the representative projection of the Cloud dataset seen in Figure 14(a) as well as the randomly selected projection in Figure 14(b) both show similar structures. For instance, the two dominating clusters ● ●, various smaller bands, as well as a joint connecting an identical subset of bands exist in both projections. Furthermore, clusters are not intermingled. This is in accordance with the pairwise similarity matrix in Figure 6(a), which suggests that each projection represents the ensemble fairly well.

Nonetheless, t-SNE breaks some bands differently in every projection. When viewing only the projection in Figure 14(a), the connectivity of separated bands cannot be recovered since neighbourhoods of distant points are not considered. For instance, it cannot be recovered that clusters ● ● ● as well as ● ● are connected in most projections. If the projection in Figure 14(b) is chosen as the representative, other connections get lost, such as these of cluster ●. When using the computed clusterings for each projection—obtained as in Section 4.2—as input for the method of Kumpf *et al.*, some

of the missing links can be exposed by querying for cluster-robust subsets as seen in Figure 1(a1). Here, the point marked with a cross is queried for all points being located in the same cluster for at least 70% of all projections. All highlighted points are covered by the two red lenses. As a result, the connection between ● and ● can be identified.

Next, the same analysis is run for the MNIST dataset projected with MDS. It provides two blocks of projections, as shown in Figure 6(b). By applying the PageRank algorithm, a representative for the larger block is obtained (see Figure 1c), which is compared to a randomly selected projection of each block. Figure 14(c) clearly shows that the projection selected from the larger block is a rotated version of the representative, while many clusters are intermingled when viewing the projection of the smaller block shown in Figure 14(d). We make this observation for all projections we randomly sampled from both blocks, indicating that the selected representative indeed represents the larger block of the dataset.

6. Conclusion and Future Work

We have proposed a new approach for visualizing the stability of k -neighbourhoods in ensembles of 2D point sets. We have demonstrated the use of this approach for identifying coherent subgroups of points over many ensemble members, and thus, assessing the robustness of dimensionality reduction techniques. For this purpose, we have introduced a novel stability measure for k -order Voronoi diagrams, and we have used this measure to determine stable neighbourhoods over many point sets. The similarity measure has also been applied to determine a subset of representative ensemble members using matrix seriation techniques as well as the PageRank algorithm. In comparison to the cluster-driven analysis of point sets, the proposed approach does not rely on the choice of an intermediate clustering and succeeds in detecting inter-cluster details and intra-cluster relationships.

In a number of examples, we have shown that the proposed approach gives interesting insights into the behaviour of dimensionality reduction techniques. Besides quantifying the sensitivity of techniques to variations in their initial parametrizations, certain characteristics of the used techniques could be

revealed. Furthermore, the proposed approach has revealed structures in datasets that were previously unknown.

In future work, we will address the following issues: Firstly, we will investigate whether the proposed similarity measure can be used to progressively find the most representative ensemble members, without the need to compute many of them in advance. Secondly, we will examine performance improvements to enable interactive insertion and re-positioning of points. In this way, we hope to gain further insight into the particular strengths and weaknesses of dimensionality reduction techniques, and eventually provide data-specific guidelines for their use. Lastly, we intend to apply the proposed technique in other fields such as crowd analysis and fluid dynamics. We see in particular the application to particle sets which are seeded in a certain region, to determine subgroups that remain stably together during particle integration.

Acknowledgements

The research leading to these results has been done within the subprojects B5 ‘Data-driven ensemble visualization’ of the Transregional Collaborative Research Center SFB/TRR 165 ‘Waves to Weather’ funded by the German Research Foundation (DFG). We thank all the reviewers for their constructive criticism and valuable comments.

Open access funding enabled and organized by Projekt DEAL. [Correction added on 26 October 2020, after first online publication: Projekt Deal funding statement has been added.]

References

- [ASE16] ATHAWALE T., SAKHAE E., ENTEZARI A.: Isosurface visualization of data with nonparametric models for uncertainty. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (January 2016), 777–786.
- [Aup07] AUPETIT M.: Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing* 70, 7 (2007), 1304–1330. *Advances in Computational Intelligence and Learning*.
- [BBHR*16] BEHRISCH M., BACH B., HENRY RICHE N., SCHRECK T., FEKETE J.-D.: Matrix reordering methods for table and network visualization. *Computer Graphics Forum* 35, 3 (2016), 693–716.
- [BD05] BALZER M., DEUSSEN O.: Voronoi treemaps. In *IEEE Symposium on Information Visualization, 2005 (INFOVIS 2005)* (October 2005), pp. 49–56.
- [BDH96] BARBER C. B., DOBKIN D. P., HUHDANPAA H.: The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software* 22, 4 (1996), 469–483.
- [BM10] BRUCKNER S., MOLLER T.: Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1468–1476.
- [BP98] BRIN S., PAGE L.: The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems* 30, 1 (1998), 107–117. *Proceedings of the Seventh International World Wide Web Conference*.
- [BvLBS11] BREMM S., VON LANDESBERGER T., BERNARD J., SCHRECK T.: Assisted descriptor selection based on visual comparative data analysis. *Computer Graphics Forum* 30, 3 (2011), 891–900.
- [CC00] COX T. F., COX M. A.: *Multidimensional Scaling*. Chapman and Hall/CRC, London, 2000.
- [Che02] CHEN C.-H.: Generalized association plots: Information visualization via iteratively generated correlation matrices. *Statistica Sinica* 12, 1 (2002), 7–29.
- [CMS13] CAMPELLO R. J. G. B., MOULAVI D., SANDER J.: Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining* (Berlin, Heidelberg, 2013), J. PEI, V. S. TSENG, L. CAO, H. MOTODA and G. XU (Eds.), Springer, Berlin, Heidelberg, pp. 160–172.
- [DJW16] DEMIR I., JAREMA M., WESTERMANN R.: Visualizing the central tendency of ensembles of shapes. In *SIGGRAPH ASIA 2016 Symposium on Visualization* (2016), ACM.
- [EH08] EDELSBRUNNER H., HARER J.: Persistent homology — A survey. *Contemporary Mathematics* 453 (2008), 257–282.
- [EH10] EDELSBRUNNER H., HARER J.: *Computational Topology: An Introduction*. American Mathematical Society, Providence, RI, 2010.
- [ELLS11] EVERITT B. S., LANDAU S., LEESE M., STAHL D.: *Cluster Analysis*. John Wiley & Sons, Ltd, West Sussex, 2011.
- [FBW16] FERSTL F., BÜRGER K., WESTERMANN R.: Streamline variability plots for characterizing the uncertainty in vector field ensembles. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (January 2016), 767–776.
- [FFST19] FAVELIER G., FARAJ N., SUMMA B., TIERNY J.: Persistence atlas for critical point variability in ensembles. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (January 2019), 1152–1162.
- [FKRW16] FERSTL F., KANZLER M., RAUTENHAUS M., WESTERMANN R.: Visual analysis of spatial variability and global correlations in ensembles of Iso-Contours. *Computer Graphics Forum* 35, 3 (2016).
- [FT74] FRIEDMAN J. H., TUKEY J. W.: A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers* C-23, 9 (September 1974), 881–890.
- [HA85] HUBERT L., ARABIE P.: Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.
- [HOGJ13] HUMMEL M., OBERMAIER H., GARTH C., JOY K.: Comparative visual analysis of Lagrangian transport in CFD ensembles. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (December 2013), 2743–2752.

- [JDKW15] JAREMA M., DEMIR I., KEHRER J., WESTERMANN R.: Comparative visual analysis of vector field ensembles. In *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)* (October 2015), pp. 81–88.
- [JHB*17] JÄCKLE D., HUND M., BEHRISCH M., KEIM D. A., SCHRECK T.: Pattern trails : Visual analysis of pattern transitions in subspaces. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2017).
- [KH13] KEHRER J., HAUSER H.: Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics* 19, 3 (March 2013), 495–513.
- [KTB*18] KUMPF A., TOST B., BAUMGART M., RIEMER M., WESTERMANN R., RAUTENHAUS M.: Visualizing confidence in cluster-based ensemble weather forecast analyses. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 109–119.
- [KW78] KRUSKAL J. B., WISH M.: *Multidimensional Scaling. Number 07–011 in Sage University Paper series on quantitative applications in the social sciences*. Sage Publications, Beverly Hills, 1978.
- [LA11] LESPINATS S., AUPETIT M.: Checkviz: Sanity check and topological clues for linear and non-linear mappings. *Computer Graphics Forum* 30, 1 (2011), 113–125.
- [LBBH98] LECUN Y., BOTTOU L., BENGIO Y., HAFFNER P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (November 1998), 2278–2324.
- [Lee82] LEE D.-T.: On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computers C-31*, 6 (June 1982), 478–487.
- [LMW*16] LIU S., MALJOVEC D., WANG B., BREMER P.-T., PASCUCCI V.: Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (2017), 1249–1268.
- [LPK05] LOVE A. L., PANG A., KAO D. L.: Visualizing spatial multivalued data. *IEEE Computer Graphics and Applications* 25, 3 (May 2005), 69–79.
- [LSP*10] LEX A., STREIT M., PARTL C., KASHOFER K., SCHMALSTIEG D.: Comparative analysis of multidimensional, quantitative data. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (November 2010), 1027–1035.
- [LT16] LEHMANN D. J., THEISEL H.: Optimal sets of projections of high-dimensional data. *IEEE Transactions on Visualization & Computer Graphics* 22, 1 (January 2016), 609–618.
- [MWK14] MIRZARGAR M., WHITAKER R., KIRBY R. M.: Curve boxplot: Generalization of boxplot for ensembles of curves. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2654–2663.
- [NSB04] NOCKE T., SCHUMANN H., BÖHM U.: Methods for the visualization of clustered climate data. *Computational Statistics* 19, 1 (February 2004), 75–94.
- [OLK*14] OELTZE S., LEHMANN D. J., KUHN A., JANIGA G., THEISEL H., PREIM B.: Blood flow clustering and applications in virtual stenting of intracranial aneurysms. *IEEE Transactions on Visualization and Computer Graphics* 20, 5 (2014), 686–701.
- [PFMA06] PINHO R., FERREIRA DE OLIVEIRA M. C., MINGHIM R., ANDRADE M. G.: Voromap: A voronoi-based tool for visual exploration of multi-dimensional data. In *10th International Conference on Information Visualisation (IV'06)* (July 2006), pp. 39–44.
- [PH13] PÖTHKOW K., HEGE H.-C.: Nonparametric models for uncertainty visualization. *Computer Graphics Forum* 32, 3 (2013), 131–140.
- [PWB*09] POTTER K., WILSON A., BREMER P.-T., WILLIAMS D., DOUTRIAUX C., PASCUCCI V., JOHNSON C. R.: Ensemble-Vis: A framework for the statistical visualization of ensemble data. In *Proceedings of the IEEE International Conference on Data Mining Workshops* (2009), pp. 233–240.
- [SG02] STREHL A., GHOSH J.: Cluster ensembles—A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research* 3 (December 2002), 583–617.
- [SVM14] SORZANO C. O. S., VARGAS J., MONTANO A. P.: A survey of dimensionality reduction techniques. arXiv preprint arXiv:1403.2877, 2014.
- [SZD*10] SANYAL J., ZHANG S., DYER J., MERCER A., AMBURN P., MOORHEAD R. J.: Noodles: A tool for visualization of numerical weather model ensemble uncertainty. *IEEE Transactions on Visualization and Computer Graphics* 16 (2010), 1421–1430.
- [TGH*16] THYNG K. M., GREENE C. A., HETLAND R. D., ZIMMERLE H. M., DiMARCO S. F.: True colors of oceanography: Guidelines for effective and accurate colormap selection. *Oceanography* 29 (September 2016). <https://doi.org/10.5670/oceanog.2016.66>.
- [TN14] THOMAS D. M., NATARAJAN V.: Multiscale symmetry detection in scalar fields by clustering contours. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2427–2436.
- [VDM14] VAN DER MAATEN L.: Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research* 15, 1 (2014), 3221–3245.
- [vH08] VAN DER MAATEN L., HINTON G.: Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research* 9 (November 2008), 2579–2605.
- [WBJ*18] WELLMANN C., BARRETT A. I., JOHNSON J. S., KUNZ M., VOGEL B., CARSLAW K. S., HOOSE C.: Using emulators to understand the sensitivity of deep convective clouds and hail to

- environmental conditions. *Journal of Advances in Modeling Earth Systems* 10 (2018). <https://doi.org/10.1029/2018MS001465>.
- [WCR*17] WENSKOVITCH J., CRANDELL I., RAMAKRISHNAN N., HOUSE L., LEMAN S., NORTH C.: Towards a systematic combination of dimension reduction and clustering in visual analytics. *IEEE Transactions on Visualization and Computer Graphics* 24 (2018), 131–141.
- [WHL18] WANG J., HAZARIKA S., LI C., SHEN H.: Visualization and visual analysis of ensemble data: A survey. *IEEE Transactions on Visualization and Computer Graphics* 25 (2019), 2853–2872.
- [Wil11] WILKS D. S.: *Statistical Methods in the Atmospheric Sciences*. Academic Press, Waltham, MA, 2011.
- [WMK13] WHITAKER R. T., MIRZARGAR M., KIRBY R. M.: Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2713–2722.



Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Video of varying k - Example a

Video of varying k - Example b

Learning Generic Local Shape Properties for Adaptive Super-Sampling

Christian Reinbold  and Rüdiger Westermann 

Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany

Abstract

We propose a novel encoder/decoder-based neural network architecture that learns view-dependent shape and appearance of geometry represented by voxel representations. Since the network is trained on local geometry patches, it generalizes to arbitrary models. A geometry model is first encoded into a sparse voxel octree of features learned by a network, and this model representation can then be decoded by another network in-turn for the intended task. We utilize the network for adaptive super-sampling in ray-tracing, to predict super-sampling patterns when seeing coarse-scale geometry. We discuss and evaluate the proposed network design, and demonstrate that the decoder network is compact and can be integrated seamlessly into on-chip ray-tracing kernels. We compare the results to previous screen-space super-sampling strategies as well as non-network-based world-space approaches.

1. Introduction

Scene representation networks (SRNs) have gained popularity for single- and multi-view object reconstruction [LGZL*20, MST*20, NMOG20, PFS*19, SZW19, YKM*20] and realtime rendering [GKJ*21, TLY*21]. SRNs represent models by features that are learnt by a network, so called latent codes. They can be accessed at arbitrary positions to obtain model-specific information such as shortest distance to the model or color. The networks overfit to the model they are trained for and do not generalize to other models. For shape reconstruction from 3D point clouds, Jiang et al. address this limitation by encoding implicit functions of local geometry patches in low dimensional latent codes, and optimizing for those during reconstruction [JSM*20]. Similarly, local geometric detail can be encoded in 3D style codes and transferred to coarse geometric representations for geometry upscaling [CKF*21], and to implement data-driven mesh subdivision [LKC*20].

We extend on current SRNs as follows: We present a novel encoder/decoder-based network architecture, called PatchNet, which learns generic local properties of geometric shapes on different levels of details (LoDs). The architecture generalizes to new models by learning a view-dependent encoding of local geometry patches, so that at a coarse scale the network can predict the appearance of geometric details. The learned feature representations are organized in a hierarchical LoD data structure to support coarse to fine look-ahead at various scales.

We utilize the aforementioned capabilities in ray-tracing, to predict super-sampling patterns when seeing coarse-scale geometry. Therefore, we ray-trace against a Sparse Voxel Octree (SVO) [LK11], where each node represents a voxel that intersects the ge-

ometry and stores averaged luminance L_V and normal N_V over the intersected surface. For each pixel, a ray is intersected with the SVO by traversing it in top-down manner until the projected voxel size is approximately equal to the pixel size. Then, the pixel appearance can be estimated by L_V at the intersected voxel. If the local geometry shows self-occlusions for the current view, or projects only to a sub-pixel area, L_V is likely to be a bad approximation. Super-sampling patterns attempt to identify this case and compute improved estimates by tracing several rays for each critical pixel.

We compare our approach to a non-network-based world-space approach as well as classical [LRU85, Mit87, RFS03a, RFS03b, XSSZ07] and network-based [KKR18, WITW20] screen-space super-sampling strategies. Conceptually, screen-space approaches refine a pixel whenever its neighboring pixels show significant variation in appearance. They fail if neighboring pixels show similar, yet consistently wrong appearances. For instance, this effect occurs at fence-like structures where the gaps between laths are not preserved at coarser scales.

2. Method

We propose a world-space approach that uses a view-dependent per-ray oracle function $f: (\mathcal{P}, \theta) \mapsto (\sigma_{gt}, L_{gt})$ to detect pixels that need refinement. Its input is a local geometry patch $\mathcal{P}(M, p)$ of M close to the intersection point p between the ray and the SVO, as well as additional rendering parameters θ . \mathcal{P} is projected into the current pixel according to θ , and the oracle returns the fraction $\sigma_{gt} \in [0, 1]$ of the pixel covered by the projection of \mathcal{P} as well as the average seen luminance $L_{gt} \in [0, 1]$. If $\sigma_{gt} \ll 1$ or $|L_{gt} - L_V| \gg 0$, the current pixel has to be refined to obtain an accurate pixel ap-

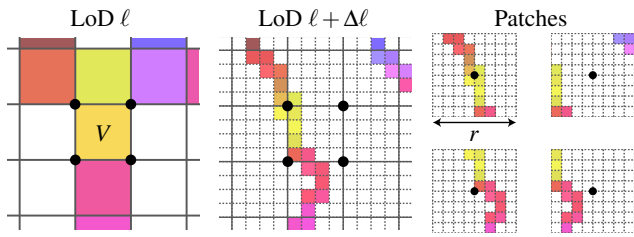


Figure 1: 2D Example showing patches for all corners of a voxel V with $\Delta\ell = 2$, $r = 8$. They are obtained by sampling blocks of fine voxels with side length r after descending $\Delta\ell$ levels of detail.

pearance. We suggest to realize f by an encoder/decoder network PatchNet, which can be evaluated efficiently on a per-pixel basis. The encoder $E: (M, p) \mapsto \mathcal{P}_{\text{enc}}(M, p)$ extracts the local geometry patch \mathcal{P} and transforms it to a low dimensional feature code \mathcal{P}_{enc} . The decoder $D: (\mathcal{P}_{\text{enc}}, \theta) \mapsto (\sigma_{\text{N}}, L_{\text{N}})$ acts as a lightweight neural renderer to compute approximate values $(\sigma_{\text{N}}, L_{\text{N}})$ for $(\sigma_{\text{gt}}, L_{\text{gt}})$.

2.1. Encoder

The encoder design is inspired by the work of Takikawa et al. [TLY*21], in which a collection \mathcal{Z} of trainable feature codes is structured in a SVO that corresponds to the geometric model. Each allocated voxel stores a d -dimensional feature code at each of its corners, with the features being shared at common corners of adjacent voxels. When SVO traversal stops by intersecting a voxel of LoD $\ell \in \mathbb{N}$ at $p \in \mathbb{R}^3$, the eight feature codes of the intersected voxel are trilinearly interpolated according to the relative position of p in the voxel. The interpolated d -dimensional vector forms the output \mathcal{P}_{enc} of the encoder. In a number of experiments we have found that $d = 10$ is sufficient to learn model independent features.

Patch Codes

In current scene representation networks, the feature codes in \mathcal{Z} are learned on a per model basis, i. e. a training process is required once per model. Instead of optimizing for the feature codes directly, we propose to train a Patch Encoder network E_{patch} that maps small patches of local geometry data to patch dependent feature codes, which we call *patch codes*. Given a corner $C \in \mathbb{R}^3$ of a voxel V at LoD ℓ in the SVO, its corresponding patch holds geometry data over a block of r^3 "fine" voxels at LoD $\ell + \Delta\ell$ that is centered at C (see Fig. 1 for a 2D example). Geometry information of each fine voxel is represented by a 5D vector with the first component encoding shape by setting it to 1 if the fine voxel is existing, and the remaining components containing L_V, N_V as stored in the fine voxel. The resulting 4D tensor of shape $(r, r, r, 5) \in \mathbb{R}^{r \times r \times r \times 5}$ is passed to $E_{\text{patch}}: \mathbb{R}^{r \times r \times r \times 5} \rightarrow \mathbb{R}^d$ to compute the patch code for C .

We found that E_{patch} can be efficiently realized by flattening the input to a $5r^3$ -dimensional vector and processing it in a multilayer network with three hidden layers of 256 channels each. Each layer is realized by taking the input vector x (or output of the previous layer) and transforming it to the output $y = \max(Mx + b, 0) \in \mathbb{R}^{256}$ with an appropriately sized matrix M and bias vector b containing learnable network weights. Experiments with 3D convolutions and

Voxelception blocks [BLRW16] increased computation costs significantly with no significant effect on predictive power. We attribute this to the oracle function f lacking translation invariance.

We set $\Delta\ell = 2$, so that PatchNet is able to "look ahead" two additional levels of detail when deciding if refined sampling is required. The block resolution is set to $r = 2 \cdot 2^{\Delta\ell}$, yielding a block of twice the side length than that of V . Thus, the network is able to perceive all geometry that can influence the current pixel, even if the pixel's center ray just barely scratches V .

2.2. Decoder

The decoder D is realized by a small multilayer network that combines the feature code of the encoder and additional rendering parameters to form the output $(\sigma_{\text{N}}, L_{\text{N}})$. The following input is provided to the decoder: a) The d -dimensional encoder output \mathcal{P}_{enc} . b) The 6D ray frame for the current pixel, i. e., the ray direction and its up vector (the projection of the camera's up vector to the orthogonal complement plane of the ray). c) The 1D pixel-voxel-ratio encoding the relative size of the projected voxel to the current pixel. It is obtained by dividing the side length of the pixel by the side length of the voxel projected onto the screen. d) All evaluations of the 25D spherical harmonics up to degree 4 at the ray direction, to facilitate learning view-dependent features. 9 out of 10 dimensions of the patch code \mathcal{P}_{enc} are interpreted as spherical harmonic coefficients by multiplying them with the nine spherical harmonic base functions up to degree 2 evaluated at the ray direction.

The decoder consists of three layers with 48 channels each. Random dropout of network units to prevent overfitting was not performed, and normalization of activation values to the same scale didn't show any improvement. The output layer is followed by a shifted and rescaled non-linear sigmoid activation function to limit the network outputs to the interval $[-1, 2]$. Outputs are further clamped to $[0, 1]$ during inference, yet no additional clamping is performed during training to avoid vanishing gradients.

2.3. Training

E_{patch} and the decoder *once* are trained on randomly sampled geometry patches of a complex geometry model. A training sample is generated by first sampling a voxel V of the model's SVO, extracting a patch at LoD $\ell + \Delta\ell$ with resolution $(3 \cdot 2^{\Delta\ell})^3$ centered at V , and rendering it into an orthographic 64×64 viewport representing a single pixel. The pixel-voxel-ratio is uniformly sampled from $[\frac{1}{2}, 1]$ and determines the size of the viewport in world space. Camera parameters are obtained via rejection sampling such that the viewport's center ray intersects V . By choosing a patch resolution as stated above, it is ensured that the patch can fill out the whole viewport independently of where the center ray intersects V .

Each training sample is processed by invoking E_{patch} at each corner of V and retrieving \mathcal{P}_{enc} at the intersection point as described in Sec. 2.1. Next, the rendering parameters are derived from the training sample and the decoder is invoked to compute $(\sigma_{\text{N}}, L_{\text{N}})$. To train the network, its weights are repeatedly updated through a gradient descent optimizer that minimizes $|\sigma_{\text{N}} - \sigma_{\text{gt}}| + |L_{\text{N}} - L_{\text{gt}}|$. The gradients are computed by back-propagating them via the chain

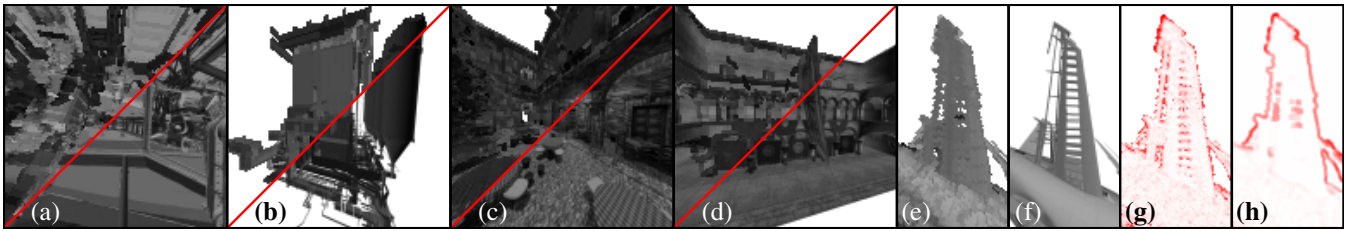


Figure 2: (a-d) Sample rendering splits (top left = 1^2 -spp, bottom right = 32^2 -spp) of all datasets used during training and evaluation. (a) Underbelly interior of Airplane, (b) Power Plant, (c) San Miguel, (d) Sponza. (e) 1^2 - and (f) 32^2 -spp renderings of the airplane's tail fin and refinement pattern as returned by (g) PN-Airplane and (h) SS-Variance.

rule through the network. The network is trained for 8 epochs on a dataset of 16 million samples, starting with a learning rate of 0.001 that is reduced by a factor of 0.1 after 4 and 7 epochs respectively.

2.4. Preprocessing & Rendering

In order to support realtime-performance during rendering, the per-voxel corner patch codes are precomputed and stored for all SVOs that are about to be rendered. This is performed by traversing each SVO and applying E_{patch} to the patches of all allocated voxel corners. Note that since E_{patch} only processes small patches of geometry, it can be readily applied to new models without costly retraining. The patch codes \mathcal{Z} are baked into the SVO, and at each intersection the codes at the corners of the intersected voxels are read.

3. Results

We evaluate PatchNet on four scenes (see Fig. 2a-d): the San Miguel scene, a bisected Sponza scene, and detailed CAD models of an airplane and power plant. Except for San Miguel, direct, diffuse lighting from two antipodal directional light sources is baked into voxel luminance. For San Miguel, we remove most plants and bake global illumination effects with the Cycles renderer of Blender. Additionally, camera tracks of 2000 frames per scene are rendered in a resolution of 130^2 pixels. All tracks start far away from geometry, close in over 1000 frames and then explore the scene's detailed geometry for another 1000 frames.

Different refinement strategies are compared by fixing a per frame budget of $X\%$ of active pixels—i. e. pixels that intersect geometry or have intersecting neighbors—that may be refined at least once. Refinement quality is measured by computing the MSE over all active pixels in any frame, with refined pixel errors being zero and unrefined pixel errors being the difference between rendering with 1^2 -spp and 32^2 -spp. Evaluations with other image metrics such as PSNR, SSIM and LPIPS yield similar results.

Fig. 3 shows the MSE values for each camera track when varying the per frame budget between 0% and 100%. Our PN-[Model] strategies are implemented by evaluating PatchNet, trained on patches of [Model], for each pixel intersecting the SVO. Pixels with high values of $|1 - \sigma_N| + \lambda \cdot |L_V - L_N|$, where $\lambda = 5$, are refined first. We compare against the—according to our findings—best performing classical screen-space based strategy SS-Variance [LRU85] that refines pixels showing a high variation of luminance in their surrounding 3×3 kernels first, as well as a world-space based strategy

WS-LUT that—for each pixel—looks up an error estimate based on the child mask of the intersected voxel. View-dependent error estimates are precomputed by rendering allocated child voxels from many views, and then stored as nine spherical harmonics coefficients for each of the 2^8 possible child configurations. We consider WS-LUT as a classical analog of our approach in which PatchNet is exchanged by a look up table. Both, SS-Variance and WS-LUT are implemented by us. Further, we evaluate against pretrained screen-space networks of Weiss et al. [WITW20] and Kuznetsov et al. [KKR18]. Both networks learn an intermediate sampling map from a low quality rendering and utilize new samples drawn according to the sampling map to enhance the results of a subsequent reconstruction/denoising pass. We feed in the 1^2 -spp rendering (plus G-buffers) and utilize the sampling map to decide which active pixels are refined first. Note that compared to the original approach by Weiss et al., we disable temporal coherence due to the lack of a flow field, and downsample the 4^2 -super-sampled sampling map by summing over 4×4 pixel blocks. Lastly, we also plot the optimal refinement strategy that assumes knowledge of the 32^2 -spp rendering and ranks pixels w.r.t. their deviation to the 1^2 -spp rendering.

The results show that our approach yields superior sampling patterns compared to Weiss et al. [WITW20] and Kuznetsov et al. [KKR18]. However, this is not entirely surprising, since their sampling masks were trained for different rendering algorithms. When refining at least 13% of active pixels, PatchNet performs consistently better than all reference methods, going as far as yielding half the MSE at 30% pixels for Airplane when comparing against SS-Variance. Further, except for San Miguel, PatchNet performs almost equally well independently of the dataset it was trained on. That is, PatchNet generalizes to other models. For San Miguel, training on other datasets yields notable worse results, although still being better than SS-Variance. We argue that San Miguel is the most complex dataset of all, including rich geometry, texture and advanced lighting. Hence, one can generalize *from* San Miguel, but not necessarily *towards* San Miguel.

Fig. 2e-h shows that PatchNet can refine porous geometry such as the airplane's tail fin. However, SS-Variance fails to detect the fine grid structure as it is lost in the coarse voxel representation rendered to screen. Similarly, as shown in Fig. 4, it cannot detect erroneous renderings of table surfaces in the San Miguel scene, where the unlit tabletop bleeds dark color into the bright tablecloth it is covered by. PatchNet detects the errors up to a certain distance, depending on the look-ahead parameter $\Delta \ell$ of PatchNet.

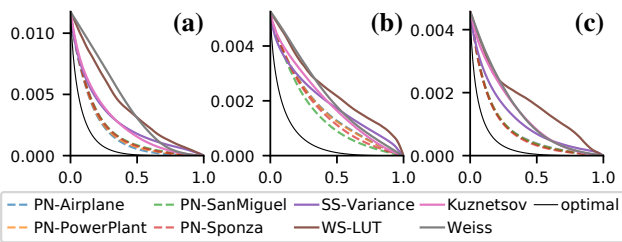


Figure 3: Evaluations on (a) Airplane, (b) San Miguel and (c) Sponza showing refinement quality in MSE for various refinement strategies plotted over per frame budgets from 0% to 100%. Results on Power Plant are similar to Airplane. Lower means better.



Figure 4: San Miguel courtyard rendered at various distances from close (block 1) to far (block 3). First image in block indicates the actual error between 1²-spp and 32²-spp renderings. Other images in block depict refinement patterns (redder areas are refined first) of strategies that fail (Image 2) or succeed (Image 3) in detecting errors at tablecloths. SS-Var stands for SS-Variance. PN- Δx represents PatchNet trained with a look-ahead of $\Delta \ell = x$.

4. Conclusion

We have presented an encoder/decoder network PatchNet that learns model independent patch codes to predict super-sampling patterns from coarse-scale geometry. Our experiments have shown that patterns predicted by our architecture are significantly more effective than those of reference methods, in particular for medium to high refinement counts, and that PatchNet generalizes to models not seen during training.

In the future, we intend to reduce inference timings during rendering by storing and traversing the SVO on the GPU, as well as evaluating the decoder in shared GPU memory. Nonetheless, we expect that inference timings will be inferior to brute-force 4x-super-sampling when casting primary rays only. As a next step, we intend to utilize PatchNet in recursive ray tracing applications, where samples are far more costly to acquire. This makes brute-force super-sampling less attractive than invoking PatchNet to identify dispensable samples. Lastly, we plan to apply model independent patch codes for model compression, learning spatially varying BSSRDF models, and predicting cone split events at rough and/or pointy surfaces in differential cone-tracing.

References

[BLRW16] BROCK A., LIM T., RITCHIE J. M., WESTON N.: Generative and Discriminative Voxel Modeling with Convolutional Neural Networks. [arXiv:1608.04236v2](https://arxiv.org/abs/1608.04236v2). 2

[CKF*21] CHEN Z., KIM V. G., FISHER M., AIGERMAN N., ZHANG H., CHAUDHURI S.: DECOR-GAN: 3D Shape Detailization by Conditional Refinement. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 15735–15744. 1

[GKJ*21] GARBIN S. J., KOWALSKI M., JOHNSON M., SHOTTON J., VALENTIN J.: FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 14346–14355. 1

[JSM*20] JIANG C., SUD A., MAKADIA A., HUANG J., NIESSNER M., FUNKHOUSER T.: Local Implicit Grid Representations for 3D Scenes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 6000–6009. 1

[KKR18] KUZNETSOV A., KALANTARI N. K., RAMAMOORTHI R.: Deep Adaptive Sampling for Low Sample Count Rendering. *Computer Graphics Forum* 37, 4 (2018), 35–44. 1, 3

[LGZL*20] LIU L., GU J., ZAW LIN K., CHUA T.-S., THEOBALT C.: Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems* (2020), vol. 33, pp. 15651–15663. 1

[LK11] LAINE S., KARRAS T.: Efficient Sparse Voxel Octrees. *IEEE Transactions on Visualization and Computer Graphics* 17, 8 (2011), 1048–1059. 1

[LKC*20] LIU H.-T. D., KIM V. G., CHAUDHURI S., AIGERMAN N., JACOBSON A.: Neural Subdivision. *ACM Trans. Graph.* 39, 4 (2020). 1

[LRU85] LEE M. E., REDNER R. A., USELTON S. P.: Statistically Optimized Sampling for Distributed Ray Tracing. *SIGGRAPH Comput. Graph.* 19, 3 (1985), 61–68. 1, 3

[Mit87] MITCHELL D. P.: Generating Antialiased Images at Low Sampling Densities. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987), SIGGRAPH '87, pp. 65–72. 1

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Computer Vision – ECCV 2020* (2020), pp. 405–421. 1

[NMOG20] NIEMEYER M., MESCHEDER L., OECHSLE M., GEIGER A.: Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020). 1

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 1

[RFS03a] RIGAU J., FEIXAS M., SBERT M.: Entropy-based Adaptive Sampling. In *Graphics Interface* (2003), vol. 2, pp. 79–87. 1

[RFS03b] RIGAU J., FEIXAS M., SBERT M.: Refinement Criteria Based on f-Divergences. In *Proceedings of the 14th Eurographics Workshop on Rendering* (2003), EGRW '03, pp. 260–269. 1

[SZW19] SITZMANN V., ZOLLHOEFER M., WETZSTEIN G.: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Advances in Neural Information Processing Systems* (2019), vol. 32. 1

[TLY*21] TAKIKAWA T., LITALIEN J., YIN K., KREIS K., LOOP C., NOWROUZSAHRAI D., JACOBSON A., MCGUIRE M., FIDLER S.: Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. In *CVPR* (2021), pp. 11353–11362. 1, 2

[WITW20] WEISS S., IŞIK M., THIES J., WESTERMANN R.: Learning Adaptive Sampling and Reconstruction for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2020). 1, 3

[XSXZ07] XU Q., SBERT M., XING L., ZHANG J.: A Novel Adaptive Sampling by Tsallis Entropy. In *Computer Graphics, Imaging and Visualization* (2007), pp. 5–10. 1

[YKM*20] YARIV L., KASTEN Y., MORAN D., GALUN M., ATZMON M., RONEN B., LIPMAN Y.: Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. In *Advances in Neural Information Processing Systems* (2020), vol. 33, pp. 2492–2502. 1

Inverting the Feature Visualization Process for Feedforward Neural Networks

Christian Reinbold¹ Rüdiger Westermann¹

Abstract

This work sheds light on the invertibility of feature visualization in neural networks. Since the input that is generated by feature visualization using activation maximization does, in general, not yield the feature objective it was optimized for, we investigate optimizing for the feature objective that yields this input. Given the objective function used in activation maximization that measures how closely a given input resembles the feature objective, we exploit that the gradient of this function w.r.t. inputs is—up to a scaling factor—linear in the objective. This observation is used to find the optimal feature objective via computing a closed form solution that minimizes the gradient. By means of Inverse Feature Visualization, we intend to provide an alternative view on a networks sensitivity to certain inputs that considers feature objectives rather than activations.

1. Introduction

To better understand the learning behavior of neural networks, the similarity of representations learned by differently trained networks has been assessed by statistical analysis of activation data (Li et al., 2015; Raghu et al., 2017). Wang et al. (2018) search for similar representations by using activation vectors and matching them over different networks. These approaches can determine similar behavior of neurons for a finite set of inputs, but they do not consider which patterns the neurons are sensitive for and, thus, neglect the semantic meaning of representations. There is also no evidence that representations behave similarly on different input sets, so that the findings are sensitive to the choice of inputs.

Another class of approaches, mainly used in image understanding tasks, tackle the problem of identifying patterns a network reacts to. For instance, pixel-wise explanations (Bach et al., 2015) and saliency maps (Simonyan et al., 2013) aim to reveal areas in input samples that certain in-

ference tasks or neurons are sensitive to. Vice versa, activation maximization (Erhan et al., 2009; Nguyen et al., 2015; Nguyen et al., 2016; Szegedy et al., 2013) or code inversion (Mahendran & Vedaldi, 2015) target the reconstruction of input samples with certain activation characteristics. The recent survey of Nguyen et al. (2019) gives a thorough overview of activation maximization approaches used in Feature Visualization (FV). Some techniques do not only analyze the activations of a single neuron, but consider groups of neurons that form a semantic unit (Olah et al., 2018). Groupings arise from the investigated network topology, i.e., convolution filters can act as semantic units of convolutional neural networks (CNNs), grouping all neurons together that share identical filter weights. Henceforth we call these units the *features* we aim to analyze, and denote by n_f the finite number of available features. To measure the features’ stimulus w.r.t. an input sample I , neuron activations of a given network N are aggregated into a single value per neuron group, yielding the input’s *feature response*—denoted by \mathbf{y}_I , a vector of dimension n_f .

In its pure form, activation maximization optimizes for an input sample \mathbf{I}^* (from the set \mathcal{I} of all valid inputs) so that $\mathbf{y}_{\mathbf{I}^*}$ resembles a prescribed vector $\mathbf{x} \in \mathbb{R}^{n_f}$, i.e.,

$$\mathbf{I}^* = \arg \max_{I \in \mathcal{I}} S(\mathbf{x}, \mathbf{y}_I), \quad (1)$$

where $S(\mathbf{x}, \mathbf{y})$ is a measure of significance of \mathbf{y} regarding \mathbf{x} . We call \mathbf{I}^* the *realization* of \mathbf{x} , and \mathbf{x} the *target objective* of \mathbf{I}^* . Maximizing for a single feature can be achieved by setting $\mathbf{x} = \mathbf{e}_i$.

Inputs stimulating a certain feature usually stimulate also many other features, yet to lesser extent. Hence, although penalized by the optimization process, feature response $\mathbf{y}_{\mathbf{I}^*}$ and target objective \mathbf{x} can differ substantially. Extending the argument that \mathbf{I}^* represents a facet of a neuron that hints towards patterns (and their granularity) it is sensitive for, we argue that an input does not necessarily represent the neurons it stimulates most, but instead should represent the neurons which cannot be stimulated more strongly by other inputs. For instance, an input I of stripes may stimulate a cross and a stripe detector neuron to equal amounts. Representation matching techniques consider both neurons equal, especially when crosses are not contained in the inputs for which activations are drawn. We argue that they represent

¹Chair of Computer Graphics and Visualization, Technical University of Munich, Bavaria, Germany.

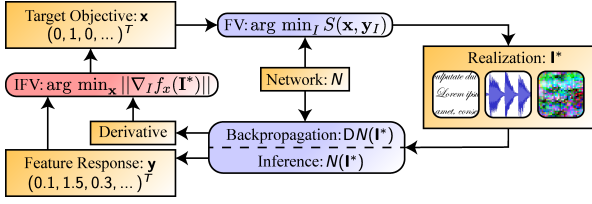


Figure 1. Interplay between Feature Visualization (FV) and Inverse Feature Visualization IFV (in red). Given an input \mathbf{I}^* that is optimized via FV for a prescribed target objective \mathbf{x} , IFV recovers \mathbf{x} solely from the network’s response to \mathbf{I}^* and gradient information. If features and neurons of the network are not in 1:1-correspondence, this is ensured via an aggregation term Agg —concatenated with the network function N —that maps activations to features. The function $f_{\mathbf{x}}$ is given by $I \mapsto S(\mathbf{x}, \mathbf{y}_I)$.

two semantically different concepts. Instead of using activations, as it is done in many approaches, we aim for a semantically richer representation in the form of inversely reproduced target objectives. That is, we suggest to consider the target objective \mathbf{x} that yields I when applying FV. Thus, stripes in the input will only be associated with stripe detectors, while cross detectors are stimulated more if stripes are exchanged with crosses.

In this work, we make a small step towards identifying target objectives. We propose a method for *Inverse Feature Visualization* (IFV) that—given a network that can be back-propagated and an input optimized via FV—can reconstruct the input’s target objective \mathbf{x} . Since the FV process neither has to be deterministic nor injective—i.e. different values for \mathbf{x} do not necessarily infer different optima \mathbf{I}^* —a rigorous definition of IFV is more complex: When seeing FV as a random variable Y over the domain \mathcal{I} with its probability density function h_Y depending on \mathbf{x} as additional parameter, IFV means to compute the maximum likelihood estimator of Y , i.e. $\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} \widehat{h}_Y(I; \mathbf{x})$ for a given input configuration I .

Since realizations \mathbf{I}^* returned by FV are locally optimal solutions of the objective function $f_{\mathbf{x}}: I \mapsto S(\mathbf{x}, \mathbf{y}_I)$ (or are at least close to them), the necessary condition for optimality $\|\nabla f_{\mathbf{x}}(I)\| = 0$ is supposed to hold. In order to recover the (most likely) target objective of a realization \mathbf{I}^* , we solve for

$$\arg \min_{\mathbf{x} \in \Omega} \|\nabla f_{\mathbf{x}}(\mathbf{I}^*)\|^2. \quad (2)$$

with Ω being the space of allowed target objectives. An overview of the principle approach underlying our work is shown in Fig. 1.

Unfortunately, solving Opt. (2) directly usually fails due to trivial solutions for \mathbf{x} . These occur at saddle or minimal points of $f_{\mathbf{x}}$, or are induced by nontrivial co-kernels of

matrices that propagate when chaining Jacobian matrices to form $\nabla f_{\mathbf{x}}$. We introduce a method called *Gradient based Inverse Feature Visualization* (Grad-IFV) to address these limitations. Our key idea is to eliminate saddle points by introducing factors in $f_{\mathbf{x}}$ and intersecting the search space Ω with an appropriate subspace of \mathbb{R}^{n_f} . The consequential reformulation of Opt. (2) is solved by computing a singular value decomposition of a matrix derived from the gradient of the objective function $f_{\mathbf{x}}$.

While in this work we demonstrate that Grad-IFV can reproduce the target objective for which a given input was optimized, building upon this observation it then needs to be investigated whether Grad-IFV can be extended to arbitrary input. In this way, it may even become possible to control certain patterns in the inputs and ask for the specific features that are sensitive for them. This sheds light on the question whether it can be determined which patterns in the given inputs are relevant and which feature combinations of a network have learned these patterns, i.e., strive toward transfer learning. By inverting an input representing feature \mathbf{x} in one network, the feature $\tilde{\mathbf{x}}$ that is represented by this input in another network can be obtained. This can give rise to a feature-based comparison of learned representations, including insight into the relevance of patterns for successful network training.

2. Method

In this work, we consider FV objective functions provided by the Lucid library (Olah et al., 2017). For a thorough discussion of the use and interpretation of objective functions in the context of FV, let us refer to the recent work by Carter et al. (2019). The objective function $f_{\mathbf{x}}$ is a concatenation of three functions N , Agg and $S_{\mathbf{x}}$, where $N: \mathcal{I} \rightarrow \mathcal{A}$ represents the network and maps an input to activations that were tapped from the network, $\text{Agg}: \mathcal{A} \rightarrow \mathbb{R}^{n_f}$ aggregates neuron activations into a *feature response* \mathbf{y} of which component i represents excitement of feature i and $S_{\mathbf{x}}: \mathbb{R}^{n_f} \rightarrow \mathbb{R}$ is defined as $S_{\mathbf{x}}(\mathbf{y}) = \mathbf{x}^T \mathbf{y} \cdot (\mathbf{x}^T \mathbf{y} / \|\mathbf{y}\|)^k$ with $k \in \mathbb{N}_0$. The set of allowed target objectives Ω is set to the $n_f D$ -sphere containing *feature directions*, i.e. normalized linear combinations of features. The term $\mathbf{x}^T \mathbf{y}$ measures the length of the projection of \mathbf{y} onto \mathbf{x} , and $\mathbf{x}^T \mathbf{y} / \|\mathbf{y}\|$ is the cosine similarity of \mathbf{x} and \mathbf{y} .

If N has an input domain \mathcal{I} that is not a vector space \mathbb{R}^m , Opt. (2) is a constrained optimization problem for which a) common solvers such as gradient descent or Adam cannot be applied, and b) the necessary condition for optimality can be violated at the domain boundary $\partial \mathcal{I}$. These issues can be circumvented by means of a differentiable, surjective mapping $P: \mathbb{R}^{n_p} \rightarrow \mathcal{I}$, which parametrizes the input domain \mathcal{I} by a real-valued vector space of dimension n_p . If such a P is available, the constrained optimization problem

$\arg \min_{I \in \mathcal{I}} f_{\mathbf{x}}(I)$ can be transformed to an unconstrained one over n_p variables via $\arg \min_{v \in \mathbb{R}^{n_p}} (f_{\mathbf{x}} \circ P)(v)$.

In the following, if $I \in \mathcal{I}$ is an input configuration, we use v_I to indicate a valid parameter vector that describes I , i.e., $P(v_I) = I$. Furthermore, P , N , Agg are combined into a single function $\tilde{N} := \text{Agg} \circ N \circ P$, which maps parameter vectors to feature responses. The feature response $\tilde{N}(v_I)$ of I is denoted by \mathbf{y}_I , and the Jacobian matrix $D\tilde{N}(v_I)$ is written as Dy_I .

2.1. Reformulation of $\nabla f_{\mathbf{x}}$

Given the described family of objective functions, the gradient $\nabla f_{\mathbf{x}}$ can be computed. The chain rule allows us to write $\nabla(f_{\mathbf{x}} \circ P)(v_I) = (k+1)q_I^k x^T Z_k(\mathbf{y}_I) \text{Dy}_I$, where $Z_k(\mathbf{y}) = Z_k(\mathbf{y})^T := \text{Id} - k/(k+1)\bar{\mathbf{y}}\bar{\mathbf{y}}^T$, $\bar{\mathbf{y}} := \mathbf{y}/\|\mathbf{y}\|$, $q_I := \mathbf{x}^T \bar{\mathbf{y}}_I$ (see supp. material). The scalar factor $(k+1)q_I^k$ equals zero if $\mathbf{x}^T \mathbf{y}_I = 0$. Depending on the parity of k , values of \mathbf{y}_I with $\mathbf{x}^T \mathbf{y}_I = 0$ either yield minimal or saddle points of $S_{\mathbf{x}}$. Since one is interested in maximizing $S_{\mathbf{x}}$, the factor can be safely dropped. Hence, if \mathbf{I}^* is a local maximum for $f_{\mathbf{x}}$, this implies that $\|\mathbf{x}^T Z_k(\mathbf{y}_{\mathbf{I}^*}) \text{Dy}_{\mathbf{I}^*}\| = 0$.

In practice, the investigated network may have linear relations that result in Dy_I and potentially $Z_k(\mathbf{y}_I) \text{Dy}_I$ having a nontrivial co-kernel that is similar for all $I \in \mathcal{I}$. For instance, dead neurons absorb gradients and introduce zero rows such that $e_i \in \bigcap_{I \in \mathcal{I}} \text{coker Dy}_I$ for some i . As a consequence, e_i will always be a trivial solution to $\|\mathbf{x}^T Z_k(\mathbf{y}_{\mathbf{I}^*}) \text{Dy}_{\mathbf{I}^*}\| = 0$, independently of the realization \mathbf{I}^* and its target objective. Thus, we introduce an additional constraint filtering out trivial solutions. Instead of solving Opt. (2), it is then solved

$$\arg \min_{\mathbf{x} \in \Omega} \|\mathbf{x}^T Z_k(\mathbf{y}_I) \text{Dy}_I\|^2 \quad \text{s.t.} \quad Z_k(\mathbf{y}_I) \mathbf{x} \in C \quad (3)$$

Here, $C \subseteq \mathbb{R}^{n_f}$ denotes a freely selectable subspace not dependent on I , which we call *critical space*. By setting

$$C := \left(\bigcap_{I \in \mathcal{I}} \text{coker Dy}_I \right)^\perp, \quad (4)$$

the previously described degeneracy can be avoided. Since the constraint is linear in \mathbf{x} , we can find a length preserving substitution $\mathbf{x} = U\sigma$ that reduces Opt. (3) to solving $\arg \min_{\|\sigma\|=1} \|\sigma^T U^T Z_k(\mathbf{y}_I) \text{Dy}_I\|^2$ (see supp. material). The solution (up to a sign) is given by the left-singular vector to the smallest singular value of $M := U^T Z_k(\mathbf{y}_I) \text{Dy}_I$. Thus, first a single forward and n_f backward passes are run to determine \mathbf{y}_I and Dy_I . Then, $MM^T \in \mathbb{R}^{n_f \times n_f}$ is computed. The eigenvalue decomposition of MM^T yields the desired left-singular vector of M .

Algorithm 1 Co-kernel Approximation

Input: Samples $\text{Dy}^{(1)}, \dots, \text{Dy}^{(n)}$, threshold ρ

for $i = 1$ **to** n **do**

Compute $U\Sigma V^T = \text{SVD}(\text{Dy}^{(i)})$

$S_i \leftarrow \text{span}\{U_{:,j} \mid \Sigma_{j,j} < \rho \cdot \|\text{Dy}^{(i)}\|_2\}$

end for

$L \leftarrow \{(S_1, 1), \dots, (S_n, 1)\}$

while $|L| > 1$ **do**

Remove $(U, w_U), (V, w_V)$ from L

Compute principal vectors $(p_U^1, p_V^1), \dots, (p_U^m, p_V^m)$

Drop all pairs with $\angle(p_U^j, p_V^j) > 45^\circ$

Interpolate $p_I^j \leftarrow \text{SLERP}(p_U^j, p_V^j; w_V/(w_U + w_V))$

$L \leftarrow L \cup \{\text{span}\{p_I^j \mid j \text{ not dropped}\}, w_U + w_V\}$

end while

return U^\perp with $(U, w_U) \in L$

2.2. Identifying the Critical Space

Since we do not make any assumptions about the architecture of N , the network's state (i.e., weights, biases, etc.) cannot be used to deduce C . Instead, we sample n random target objectives \mathbf{x}_i ($i \in \{1, \dots, n\}$), compute realizations \mathbf{I}_i^* by applying FV, and then try to approximate C by investigating the matrices $\text{Dy}^{(i)}$. We denote $\text{Dy}_{\mathbf{I}_i^*}$ by $\text{Dy}^{(i)}$ and $\mathbf{y}_{\mathbf{I}_i^*}$ by $\mathbf{y}^{(i)}$. In practice, the matrices $\text{Dy}^{(i)}$ will not develop clear co-kernels, due to numerical inaccuracy, incomplete training runs, or stochastic optimization, just to mention a few reasons. Hence, special care has to be taken when designing an algorithm to determine C . Instead of searching for co-kernels, we search for a subspace C of which its vector-matrix products with the sampled matrices $\text{Dy}^{(i)}$ diminish, i.e., the spectral norm quotients $\|C^T \text{Dy}^{(i)}\|_2 / \|\text{Dy}^{(i)}\|_2$ have to become small.

Algorithm 1 approximates Eq. (4). First, the co-kernels are roughly approximated by computing a subspace of left-singular vectors with reasonable small singular values $< \rho \cdot \|\text{Dy}^{(i)}\|_2$ for each sample. Afterwards, two subspaces at a time are merged until one is left over. The merging is implemented by computing the principal angles and vectors as suggested by [Knyazev & Argentati \(2002\)](#), dropping all vectors with principal angles of 45° or higher and then applying spherical linear interpolation (SLERP) to each pair of corresponding left and right principal vectors. Thereby, the interpolation parameter is given by the ratio of subspaces that already have been merged into either of the two spaces to merge. The set of newly acquired vectors form a basis of the merged space. It can be interpreted as a roughly approximated intersection of two spaces, with the exact intersection being obtained when dropping all principal vectors with principal angles not exactly 0° . Note that the result of Algorithm 1 depends both on choices for

the singular value threshold as well as the order in which subspaces are merged.

In a perfect world, solving Opt. (3) for a sample \mathbf{I}_i^* would yield a projected target objective $\tilde{\mathbf{x}}_i \in \Omega \cap Z_k(\mathbf{y}^{(i)})^{-1}C$ such that $Z_k(\mathbf{y}^{(i)})\tilde{\mathbf{x}}_i$ points into the same direction as $Z_k(\mathbf{y}^{(i)})\mathbf{x}_i$ projected onto C . It is uniquely determined by normalizing $Z_k(\mathbf{y}^{(i)})^{-1}C_M C_M^T Z_k(\mathbf{y}^{(i)})\mathbf{x}_i$. This means that the realization’s target objective \mathbf{x}_i can only be recovered modulo a shift in $(Z_k(\mathbf{y}^{(i)})^{-1}C)^\perp$ direction followed by a renormalization. One cannot expect to be any better since the objective function parametrization $\mathbf{x} \mapsto f_{\mathbf{x}}(\mathbf{I}_i^*)$ is (by construction of C) constant on $(Z_k(\mathbf{y}^{(i)})^{-1}C)^\perp$ -shifts. In terms of maximum likelihood estimation, one still finds a solution for $\arg \max_{\mathbf{x}} f_Y(\mathbf{I}; \mathbf{x})$. However, it may be another point on the plateau of the graph of $f_Y(\mathbf{I}; \cdot)$ where the realization’s target objective one started with resides as well.

2.3. The Adversary - Aggregation

Common choices for Agg [Olah et al. \(2017\)](#) either average along the activations of all neurons associated with a feature, or pick a single representative activation—for instance the center pixel of a feature map. Both approaches have limitations. When aggregating over activations returned by a convolution filter such as a Sobel filter, except for border pixels, all contributions of pixels being convolved cancel out. Any filter would thus degenerate to a detector of color patches plus some border pattern. More general, aggregating along a dimension would eliminate the effect of a preceding linear operation in this dimension. When picking only a single neuron’s activation as representative, the realization returned by FV becomes sensitive to its receptive field, rendering realizations of different layers or network architectures incomparable. Although this does not pose an immediate issue for IFV itself, it counteracts our main motivation of using IFV techniques for network comparison later on.

To overcome these limitations, we propose to use a max-pooling operation followed by mean-aggregation. While the latter operation is independent of the receptive field, the former breaks linearities and allows us to still distinguish between different convolution filters. Note that a basic mean operation may also be sufficient if the activations returned by N are generated by a rectified linear activation function (ReLU) or max-pooling operation. We suggest to always include operations breaking linearity in the aggregation process.

3. Results and Evaluation

In the following we analyze the accuracy of Grad-IFV and how the recovered target objectives differ from feature re-

sponses. FV is applied to three networks with different topology, size and application domains, to generate realizations for randomly sampled target objectives \mathbf{x}_i . Given these realizations, Opt. (3) is solved to yield the solutions \mathbf{x}_i^* , which are then compared to the target objectives \mathbf{x}_i . We will subsequently call the solutions \mathbf{x}_i^* *predicted objectives*, since they are returned by Grad-IFV and are supposed to be an accurate estimate for the target objectives.

3.1. Network Architectures

GoogLeNet: GoogLeNet ([Szegedy et al., 2015](#)) builds upon stacked Inception modules, each of which takes the input of the previous layer, applies convolution filters with kernel sizes $(1 \times 1, 3 \times 3, \text{ and } 5 \times 5)$, and concatenates all resulting feature maps into a single output vector. GoogLeNet has been trained for image classification, its input domain is given by 224×224 RGB-images.

DenseNet: The DenseNet-BC architecture ([Huang et al., 2017](#)), with growth rate $k = 12$, has been trained on the CIFAR-10 subset of the 80 million Tiny Images dataset ([Torralba et al., 2008; Krizhevsky, 2009](#)). Its input domain is given by 224×224 RGB-images that are partitioned into 10 different classes. Three dense blocks are connected via convolution layers, followed by max pooling, form the core part of the network. Within a dense block, each layer takes the outputs of *all* preceding layers as inputs and processes them by applying a 1×1 -bottleneck convolution, batch normalization, ReLU activation, and finally a 3×3 -convolution. The output is passed to all subsequent layers of the dense block. In total, 0.8M weights are trained with weight decay by stochastic gradient descent with nesterov momentum. During training, images are flipped, padded with 4 pixels on each side, and cropped back to its original size with a random center.

SRNet: The fully convolutional SRNet ([Sajjadi et al., 2018](#)) has been modified and trained to upscale low resolution geometry images (normal and depth maps) of isosurfaces in scalar volume data by [Weiss et al. \(2019\)](#). The low resolution input size is set to 128×128 pixels. Residual blocks of 3×3 convolutions transform the input maps into a latent space representation, which is then upsampled, folded, and added as residual to bilinearly upscaled versions of the input. The input domain is given by a 2D normal field comprised of 3D vectors, a 2D depth map with values in the range $[0, 1]$, and a 2D binary mask with values in $\{-1, 1\}$ to indicate surface hits during rendering.

3.2. Feature Visualization

If not stated otherwise, the components of the objective function $f_{\mathbf{x}}$ used for FV are as follows: The network function N applies the network up to the investigated layers (that is `inception4c`, `dense3` etc.) and outputs their respec-

tive activations as a 3D-tensor with 2 spatial and 1 channel dimension. The aggregation operation `Agg` then applies 2D max pooling with kernel size 3×3 and a stride of 1 followed by taking the mean. Both operations are performed along the spatial dimension so that one ends up with a 1D vector of „channel activations” that act as our feature response. Here, each channel constitutes a separate feature. The feature response is combined with the target objective \mathbf{x} as described in Sec. 2. The power k of the cosine term is set to 2. Last but not least, the used parametrization P_{rgb} yields batches of RGB images clamped to values in $[0, 1]$ by applying the sigmoid function to the unbounded space $\mathbb{R}^{H \times W \times 3}$, where H and W denote the respective input image dimensions for the investigated network.

The target objectives \mathbf{x} with nonnegative entries are sampled from a vector distribution X with uniform Hoyer sparseness measure (i.e. $\text{hoyer}(X) \sim \mathcal{U}\{0, 1\}$) (Hoyer, 2004). After sampling \mathbf{x} , we optimize for $\arg \min_{v \in \mathbb{R}^{n_p}} (f_{\mathbf{x}} \circ P)(v)$ by first applying Adam optimization (Kingma & Ba, 2014) followed by some steps of fine tuning with L-BFGS (Byrd et al., 1995). For DenseNet and SRNet, we run 800 steps of Adam and 300 steps of L-BFGS. To let GoogLeNet converge to an optimum, we run 3000 steps of Adam followed by 500 steps of L-BFGS.

3.3. Time Complexity & Performance

The runtime complexity of solving Opt. (3) and Algorithm 1 is $\mathcal{O}(\max(n_p n_f^2, n_f^3))$ per sample. Runtime is dominated by singular value decompositions and multiplication of $n_f \times n_f$ and $n_f \times n_p$ matrices. The computation of Dy requires n_f backward passes through the network, making it strongly dependent on the network architecture. Performance measurements are performed on a server architecture with 4x Intel Xeon Gold 6140 CPUs with 18 cores @ 2.30GHz each, and an NVIDIA GeForce GTX 1080 Ti graphics card with 11 GB VRAM. Timings are shown in Table 1. Note that the generation of realizations takes significantly longer due to the additional L-BFGS steps. Except for Algorithm 1, all computations are executed on the GPU.

3.4. The Simple Case

In the first experiment, the 512 filters in the `inception4c` layer of GoogLeNet, which we abbreviate by GN4c, are investigated. We sample 150 target objectives \mathbf{x}_i as described above and add further 150 canonical vectors of sparsity measure 1. Then, FV is performed. For each resulting realization \mathbf{I}_i^* , a target objective \mathbf{x}_i^* is predicted by solving Opt. (3), and then compared to \mathbf{x}_i by measuring their angular distance in degrees, i.e., $180/\pi \cdot \min\{\cos^{-1}(-\mathbf{x}_i^T \mathbf{x}_i^*), \cos^{-1}(\mathbf{x}_i^T \mathbf{x}_i^*)\}$. The elimination of trivial solutions is not required yet, hence we set $C = \mathbb{R}^{n_f}$.

Table 1. Timings for computing realizations via FV (\mathbf{I}_i^*), back-propagation ($\text{Dy}^{(i)}$), solving Opt. (3) (\mathbf{x}_i^*) and re-optimization of \mathbf{I}_i^* (re-opt.), for different number of input parameters (n_p) and features (n_f). All timings are in seconds *per sample*. Timings associated to IFV are in bold. Critical space computation for DN[4]cl takes 0.5s on average per space. Batch sizes in FV are 8 for GN4c, 64 for DN[4]cl and DNde, and 32 for SRNet. Re-optimization always is performed with a batch size of 64. Solutions of Opt. (3) are computed for all samples at once, in a single batch.

Model	n_p	n_f	\mathbf{I}_i^*	$\text{Dy}^{(i)}$	\mathbf{x}_i^*	re-opt.
GN4c	$224^2 \cdot 3$	512	17.3	0.7	0.2	0.7
DN[4]cl	$32^2 \cdot 3$	10	2.1	0.0	0.0	0.7
DNde	$32^2 \cdot 3$	342	2.0	0.3	0.1	0.7
SRNet	$128^2 \cdot 4$	64	10.7	0.4	0.0	3.1

The horizontal margin distribution in Fig. 2a shows the angular distances between the target objectives and either the predicted objectives \mathbf{x}_i^* , feature responses $\mathbf{y}^{(i)}$, or randomly sampled vectors. Note that whereas the angular distance between \mathbf{x}_i and $\mathbf{y}^{(i)}$ is about 45° (or a cosine similarity of ≈ 0.7) with large variance, the predictions \mathbf{x}_i^* show only a deviation of approx. 10° (≈ 0.98 in terms of cosine similarity!) with very few outliers.

Next, it is verified that the realizations are optimal w.r.t. the predicted objective. To achieve this, a realization is re-optimized w.r.t. either \mathbf{x}_i , \mathbf{x}_i^* , $\mathbf{y}^{(i)}$, or a random objective with Adam for 500 steps. Finally, the SSIM index between the realization before and after optimizing is computed. SSIM is a perception-based model to measure the similarity of two images w.r.t. structural information (Zhou Wang et al., 2004).

The results are shown in Fig. 2a. High SSIM indices confirm that almost all realizations remain optima under the predicted objectives, yet there is clearly more change happening when re-optimizing for the feature response as objective. This observation confirms the rationale underlying the Deep-Dream process (Mordvintsev et al., 2015) that enhancing features changes the input significantly. In particular, in many cases re-optimizing for the feature response yields similar results to re-optimizing for a random vector.

Note that the SSIM index is strictly lower than one when re-optimizing for the target objective. This observation relates back to the Adam optimizer having an internal state that has to warm up first before Adam convergences. Hence, even when initializing Adam with an instable, local optimum, Adam may leave it and converge to a completely different one, lowering the SSIM index significantly.

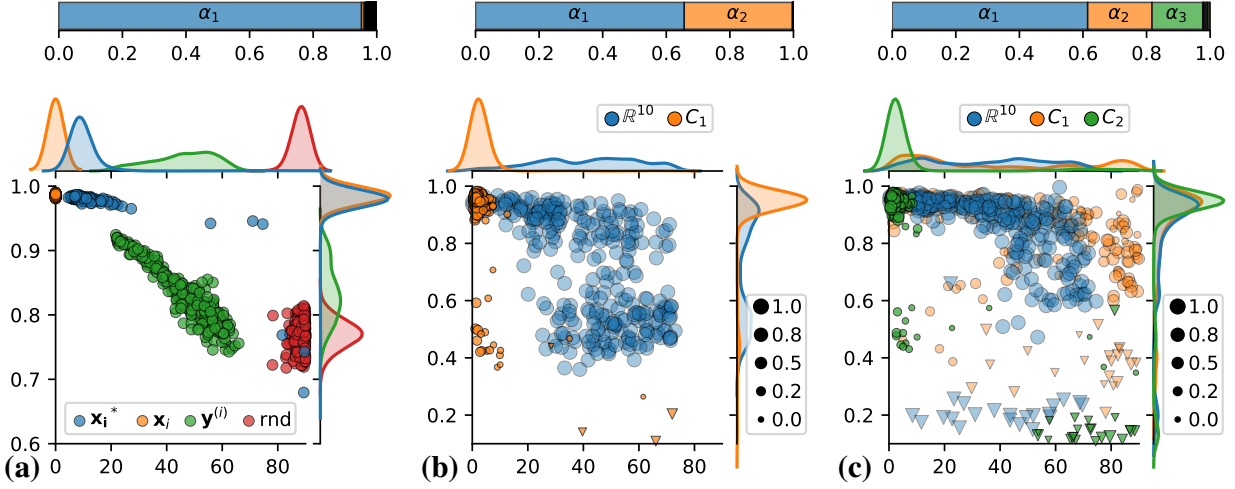


Figure 2. (a) GoogLeNet, (b) DenseNet, (c) DenseNetEx4. (top) Values $\alpha_1, \dots, \alpha_{10}$ indicating the mean contribution of j -th left-singular vectors to the target objectives, stacked in a bar chart in increasing order of j . Bars of missing values are too small for plotting. (bot (a)) Bivariate distribution of angular distances to the target objectives (horizontal axis) and SSIM indices (vertical axis) when re-optimizing w.r.t. predicted objectives, target objectives, feature responses or random feature objectives. (bot (b,c)) Trivariate distribution when re-optimizing w.r.t. predicted objectives for different critical spaces. Point size encodes the fraction of the target objective \mathbf{x}_i that lives in the respective critical space, that is $\|v\|/(\|v\| + \|w\|)$ with $\mathbf{x}_i = v + w$, $v \in Z_k(\mathbf{y}^{(i)})^{-1}C$, $w \in (Z_k(\mathbf{y}^{(i)})^{-1}C)^\perp$. Triangles represent unstable samples for which re-optimization with the target objective yields an SSIM index lower than 0.7.

3.5. Utilizing the Critical Space

Next, we investigate the ten neurons in the classification layer of DenseNet for two differently trained networks. Since aggregation along spatial dimensions becomes unnecessary in this case, we set $\text{Agg} \equiv \text{Id}$. The second network is trained in a similar way than the first network described above, yet it never sees any training data for class 4. The classification layers of both networks are denoted by DN4cl and DN4cl respectively. We sample 290 target objectives and add the ten canonical ones.

When predicting objectives and comparing them to the target objectives \mathbf{x}_i as in the previous section, Grad-IFV fails. The experiment represented by blue dots in Fig. 2b,c show that the predicted objectives do not resemble the target objectives. To see how close the target objective is to be an optimal solution for Opt. (3), we decompose each \mathbf{x}_i as follows: Let $v_{i,1}, \dots, v_{i,n_f}$ be the left-singular vectors of $Z_k(\mathbf{y}^{(i)}) \text{Dy}^{(i)}$ with singular values $\sigma_{i,1} \leq \sigma_{i,2} \leq \dots \leq \sigma_{i,n_f}$. Then we determine coefficients $c_{i,j}$ such that $\mathbf{x}_i = \sum_j c_{i,j} v_{i,j}$. Finally, we average along all squared coefficients of the same order and obtain values $\alpha_j = \sum_{i=1}^n (c_{i,j}^2/n)$ that indicate how much the j -th left-singular vectors contribute to the \mathbf{x}_i s. Note that $c_{i,0} = \mathbf{x}_i^T \mathbf{x}_i^*$ and that $90(1 - c_{i,0}^2)$ is the polynomial of degree two approximating the angular distance between \mathbf{x}_i and \mathbf{x}_i^* with coincidence in $c_{i,0}^2 \in [0, 0.5, 1]$.

Since Grad-IFV always predicts the left-singular vector of $Z_k(\mathbf{y}) \text{Dy}$ to the lowest singular value, it performs well when $\alpha_1 \approx 1$ and $\alpha_j \approx 0$ for $j > 1$. As seen in Fig. 2a, $\alpha_1 \approx 1$ holds true for the scenario of Sec. 3.4. However, the horizontal bars of Fig. 2b,c indicate additional contributions indicated by significant values for α_2 and α_3 , respectively. When further investigating left-singular vectors, we realized that all contributions of α_1 (and α_2 for DN4cl) relate back to the same span of left-singular vectors for all samples. We consider these vectors to be trivial solutions of Opt. (3), which we intend to sort out.

To obtain the trivial solutions and their complement, the critical space, the first 32 of the 300 Jacobian matrices $\text{Dy}^{(i)}$ are passed to Algorithm 1. We apply a nested intervals technique to determine all values of $\rho \in (0, 1)$ that yield critical spaces of different dimensions. For DN4cl, a single 9-dimensional critical space C_1 is determined, of which the complement is spanned by (approximately) $(1, \dots, 1)^T \in \mathbb{R}^{10}$. The same critical space C_1 is retrieved for DN4cl, accompanied by a second one C_2 of 8 dimensions with its complement being spanned by $(1, \dots, 1)^T$ and e_4 . Presumably, $(1, \dots, 1)^T$ arises as trivial solution because the network learned to exploit the softmax translation invariance in order to improve on a regularization term on its weights. Similarly, e_4 relates back to a dead neuron of a class the network of DN4cl never has seen.

The solutions \mathbf{x}_i^* of Opt. (3) are computed by setting C to either the full 10-dimensional ambient space, C_1 , or C_2 . We

evaluate how closely the predicted objectives resemble the target objectives by projecting both onto $(Z_k(\mathbf{y}^{(i)})^{-1}C)^T$ and computing their angular distance. The distribution of angular distances, which is shown in the horizontal margin plots of Fig. 2b,c, indicate that Grad-IFV performs well for one particular critical space (C_1 for DNcl and C_2 for DN4cl). In practice, the correct one can be identified by running the described procedure for a set of known, i.e. sampled, target objectives \mathbf{x} , choosing the best performing critical space, and then solving Opt. (3) for realizations \mathbf{I}_i^* with unknown target objective.

When applying re-optimization, Fig. 2b,c indicates that all samples with high angular distance to the target objective exhibit a low SSIM index. In particular, target objectives cannot be recovered accurately for samples where FV failed to find a stable optimum (marked by triangles in Fig. 2b,c). Further, there exists a cluster of (stable) samples for which the SSIM index is low although the prediction is accurate. All samples in the cluster have in common that a notable fraction of their corresponding target objectives is located in the subspace $(Z_k(\mathbf{y}^{(i)})^{-1}C)^\perp$, which we just factored out. This observation gives evidence that, in practice, the objective function parametrization $\mathbf{x} \mapsto f_{\mathbf{x}}$ is not entirely constant for $(Z_k(\mathbf{y}^{(i)})^{-1}C)^\perp$ -shifts, as it would be if C is chosen to be the exact union of co-kernels as expressed in Eq. (4). Thus, whenever we eliminate outliers by dropping all predicted objectives with a low SSIM value, we loose some relevant results as well.

3.6. Dropping the Cosine Term

When experimenting with different numbers of features n_f and alternative aggregation functions, we observed that—as long as we keep the cosine term of the significance measure $S_{\mathbf{x}}$ with a power of $k = 2$ —Grad-IFV is quite resilient. When increasing the number of features or exchanging the aggregation function by averaging or picking as described in Sec. 2.3, we notice only slight increases in angular distances with prediction quality being similar to the results of Sec. 3.4. Except for the very few occasional outliers which can be filtered out by re-optimizing the input w.r.t. the predicted objective and investigating the SSIM index, the target objective is reliably recovered.

However, when running the experiments again with the cosine term dropped ($k = 0$), predictions may become unreliable. When only considering the first 160 convolution filters in the `dense3` block of DenseNet, which we call DNde from now on, predicted objectives still perform significantly better than just estimating \mathbf{x}_i via the feature response $\mathbf{y}^{(i)}$. The SSIM index obtained by re-optimization does not drop below 0.89. For more features, prediction quality starts to degenerate quickly. When considering more than 288 of the 342 available filters in DNde, almost all

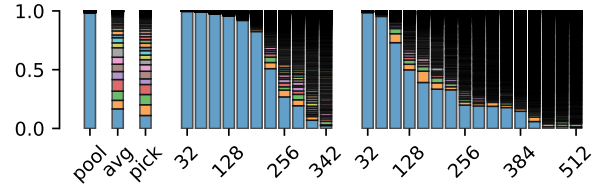


Figure 3. Values of α_j (i.e., mean contribution of j -th singular vectors, ordered by singular values, to target objectives) for different experiments with dropped cosine term, stacked in a bar chart in increasing order of j . Black indicates high fragmentation. Statistics for (left) 80 filters in the `dense3` block of DenseNet for different aggregation routines, (middle) n_f filters of DNde, and (right) GNcl, where n_f increases in steps of 32.

angular distances between \mathbf{x}_i^* s and \mathbf{x}_i s are in the range from 70° to 90° . When increasing the feature count in the `inception4c` layer of GoogLeNet (GN4c), degradation starts to set in for 96 features and continues up to 256. Beyond, most predicted objectives are perpendicular to the target objective.

As discussed in Sec. 3.5, the target objective can be expressed as a linear combination of singular vectors, to investigate by which margin Grad-IFV fails to extract the correct prediction. The resulting α_j -values for different feature counts n_f are depicted by stacked bars in Fig. 3. The prediction quality of Grad-IFV can be assessed via the height of the lowest bar, which shows the value of α_1 . High fragmentation of the bar charts suggests that target objectives cannot solely be recovered by setting up an appropriate critical space—at least not by one perpendicular to singular vectors of small singular values as returned by Algorithm 1.

Next, we consider the first 80 filters of DNde and exchange the aggregation function. The coefficient histograms of Fig. 3 show that Grad-IFV fails to extract meaningful predictions for averaging- and picking-aggregations, although the results for max pooling followed by averaging are close to perfect.

Interestingly, when the experiments are conducted with $k = 2$ —i.e. with the cosine term—accurate predictions are obtained. In this case, FV has limited options in order to produce a visualization so that objectives pointing into perpendicular direction of \mathbf{y} can be ignored during IFV. In Sec. 2.1, the matrix $Z_k(\mathbf{y})$ shrinks all vectors in the input’s feature response direction and thus makes them more likely to be the left-singular vector to the smallest singular value.

3.7. Different Parameterizations

We further investigate the influence of the parametrization P on Grad-IFV. Therefore, we define the variables v over

which FV optimizes in Fourier space and perform spatial de-correlation (Olah et al., 2017), i.e., P_{fit} is set to the inverse discrete Fourier transform (iFFT) followed by sigmoid clamping. A parametrization favoring low frequencies can be obtained by scaling Fourier coefficients according to their frequency energies before applying iFFT, denoted by P_{fite} . For each parametrization P_{rgb} , P_{fit} and P_{fite} , Grad-IFV is applied to GN4c and DNde. Although the image quality of the resulting realizations change notably (see Fig. 4), the quality of predicted objectives is not influenced. In particular, results do not suffer from low image quality of FV in case no regularizations such as transformation robustness are used.

Lastly, we analyze the SRNet for upscaling geometry images. We parameterize the normal map by coordinates (φ, θ) , yet φ is not bound to an interval length of 2π so that warping gradients at interval borders can be avoided. To ensure that the z -component is positive, we restrict θ to the interval $[-\pi/2, \pi/2]$ by applying a sigmoid function followed by an appropriate affine transformation. The depth map is clamped to $[0, 1]$ with the sigmoid function. The binary mask is either hard-coded to be one or parameterized continuously in the interval $[-1, 1]$. Note that we cannot properly represent the mask by a discrete set of values since this would turn the optimization problem of FV (Opt. (1)) into a mixed-integer programming problem.

When investigating the 64 filters of the 8th residual block of SRNet, Grad-IFV always recovers the 64 sampled target objectives accurately up to an angular distance of 5° . We even obtain reliable results with angular distances $< 20^\circ$, when dropping the cosine term of the objective function.

4. Limitations & Future Work

One of our major goals is to identify convolution filters with similar feature visualizations along different networks. Under the assumption that feature visualizations carry semantic information about what a network learns, such an approach can eventually raise network comparison from mere signal analysis of activations to a semantic level. IFV enables selecting and visualizing a feature using one network first, and then inverting the resulting realization using a second network. This yields two features obeying the same visualization and, thus, representing the same semantic concept.

In real scenarios, however, a realization that is optimal for one network will not be so for another network. It might not even be close to an optimum as long as the sensitivity of the objective function to high frequencies and noise is not reduced. In particular, Grad-IFV relies on an input that is close to optimal, since a gradient’s magnitude, in general, cannot reflect how distant an input is to an optimal solution in the surrounding.

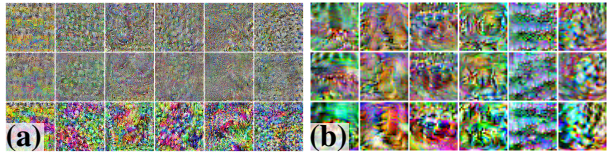


Figure 4. Realizations returned by FV when parameterizing images through (top) P_{rgb} , (mid) P_{fit} and (bot) P_{fite} . Visualized features stem from (a) GoogLeNet and (b) DenseNet.

Therefore, in future work we will consider to widen the scope of objective functions to include arbitrary input priors (such as in Mahendran & Vedaldi (2015) or Nguyen et al. (2016)). Furthermore, we intend to integrate concepts that facilitate the processing of visualizations that are robust w.r.t. transformations (Olah et al., 2017). Both, input priors and transformation robustness, are key techniques to generate consistent and interpretable visualizations. By considering such techniques in IFV, we hope to achieve feature predictions that are less sensitive to network variations or input noise.

Upon resolving these issues, IFV can be used for network comparison, to analyze learned representations of networks trained on different datasets. Here it will be interesting to investigate which features are shared between two networks, and whether invariant operations on a network’s weight space such as permuting or rescaling neurons can be recovered.

5. Conclusion

We introduce the problem of identifying target objectives under which Feature Visualization (FV) yields a certain input, and propose a solution for certain types of FV objective functions. We demonstrate that the (possibly unknown) target objective can be accurately approximated by performing a singular value decomposition of a modified version of the network’s Jacobian matrix. In cases where the Jacobian matrix is ill-behaving, we identify problematic subspaces and factor them out to obtain accurate results modulo the reduction. In a number of experiments we investigate the accuracy by which the target objective is recovered, and whether the input remains stable under the predicted objective, i.e., the result truly is an inverse of FV. We observe that different choices for layer size, aggregation and objective functions can have a significant impact on the proposed technique. Finally, we envision future research directions towards feature-based comparison of learned representations—including assessment of the relevance of patterns for successful network training—and the use of Inverse Feature Visualization for network comparison.

References

- Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise Relevance Propagation. *PLOS ONE*, 10(7):1–46, 07 2015. doi:10.1371/journal.pone.0130140.
- Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. doi:10.1137/0916069.
- Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. Activation Atlas. *Distill*, 2019. doi:10.23915/distill.00015. <https://distill.pub/2019/activation-atlas>.
- Erhan, D., Bengio, Y., Courville, A., and Vincent, P. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.
- Hoyer, P. O. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5 (Nov):1457–1469, 2004.
- Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, July 2017. doi:10.1109/CVPR.2017.243.
- Kingma, D. P. and Ba, J. Adam: A method for Stochastic Optimization, 2014. URL <https://arxiv.org/abs/1412.6980>. arXiv: 1412.6980 [cs.LG].
- Knyazev, A. V. and Argentati, M. E. Principal angles between subspaces in an A-based scalar product: Algorithms and perturbation estimates. *SIAM Journal on Scientific Computing*, 23(6):2008–2040, 2002. doi:10.1137/S1064827500377332.
- Krizhevsky, A. Learning multiple layers of features from Tiny Images. Technical report, University of Toronto, 2009.
- Li, Y., Yosinski, J., Clune, J., Lipson, H., and Hopcroft, J. Convergent Learning: Do different neural Networks learn the same representations? In Storcheus, D., Ros-tamizadeh, A., and Kumar, S. (eds.), *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pp. 196–212, Montreal, Canada, 11 Dec 2015. PMLR.
- Mahendran, A. and Vedaldi, A. Understanding deep image representations by inverting them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5188–5196, June 2015. doi:10.1109/CVPR.2015.7299155.
- Mordvintsev, A., Olah, C., and Tyka, M. Inceptionism: Going deeper into neural networks, 2015. URL <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>.
- Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436, June 2015. doi:10.1109/CVPR.2015.7298640.
- Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. Synthesizing the preferred inputs for neurons in neural networks via Deep Generator Networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3387–3395. Curran Associates, Inc., 2016.
- Nguyen, A., Yosinski, J., and Clune, J. *Understanding neural networks via Feature Visualization: A survey*, pp. 55–76. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28954-6. doi:10.1007/978-3-030-28954-6_4.
- Olah, C., Mordvintsev, A., and Schubert, L. Feature Visualization. *Distill*, 2017. doi:10.23915/distill.00007. <https://distill.pub/2017/feature-visualization>.
- Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. The Building Blocks of Interpretability. *Distill*, 2018. doi:10.23915/distill.00010. <https://distill.pub/2018/building-blocks>.
- Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6076–6085. Curran Associates, Inc., 2017.
- Sajjadi, M. S. M., Vemulapalli, R., and Brown, M. Frame-Recurrent Video Super-Resolution. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6626–6634, June 2018. doi:10.1109/CVPR.2018.00693.
- Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, 2013. URL <https://arxiv.org/abs/1312.6034>. arXiv: 1312.6034 [cs.CV].

- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2013. URL <https://arxiv.org/abs/1312.6199>. arXiv: 1312.6199 [cs.CV].
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
- Torralba, A., Fergus, R., and Freeman, W. T. 80 Million Tiny Images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, Nov 2008. ISSN 1939-3539. doi:10.1109/TPAMI.2008.128.
- Wang, L., Hu, L., Gu, J., Hu, Z., Wu, Y., He, K., and Hopcroft, J. Towards understanding Learning Representations: To what extent do different neural networks learn the same representation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 9584–9593. Curran Associates, Inc., 2018.
- Weiss, S., Chu, M., Thurey, N., and Westermann, R. Volumetric Isosurface Rendering with Deep Learning-based super-resolution. *IEEE Transactions on Visualization and Computer Graphics*, 2019. ISSN 2160-9306. doi:10.1109/TVCG.2019.2956697.
- Zhou Wang, Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image Quality Assessment: From error visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. ISSN 1057-7149. doi:10.1109/TIP.2003.819861.

Inverting the Feature Visualization Process for Feedforward Neural Networks

Supplementary Material

Christian Reinbold¹ Rüdiger Westermann¹

This document accompanies our paper *Inverting the Feature Visualization Process for Feedforward Neural Networks*. It contains additional material that could not be included in the manuscript due to page restrictions. In particular, we provide

- a mathematically rigorous formulation of the transformation of $\nabla f_{\mathbf{x}}$ (Sec. S1),
- additional details on how the target objective is sampled (Sec. S2),
- visual differences in realizations when re-optimizing,
- additional results of all performed experiments (Sec. S4).

S1. Reformulation of $\nabla f_{\mathbf{x}}$ - Calculations

In our work, we derive the gradient $\nabla f_{\mathbf{x}} \circ P$ for solving

$$\arg \min_{\mathbf{x} \in \Omega} \|\nabla(f_{\mathbf{x}} \circ P)(\mathbf{I}^*)\|^2.$$

In the following, we outline the calculations to compute the gradient. Let us recall that $f_{\mathbf{x}} \circ P = S_{\mathbf{x}} \circ \tilde{N}$, where $S_{\mathbf{x}}$ is defined as $S_{\mathbf{x}}(\mathbf{y}) = \mathbf{x}^T \mathbf{y} \cdot (\mathbf{x}^T \mathbf{y} / \|\mathbf{y}\|)^k$ for fixed $k \in \mathbb{N}_0$ and \tilde{N} is a function returning the feature response \mathbf{y}_I for an input parametrization v_I . With $\bar{\mathbf{y}} := \mathbf{y} / \|\mathbf{y}\|$ and $q(\mathbf{y}) := \mathbf{x}^T \bar{\mathbf{y}}$, one first obtains

$$\begin{aligned} \frac{\partial q(\mathbf{y})}{\partial y_i} &= \frac{\frac{\partial}{\partial y_i}(\mathbf{x}^T \mathbf{y}) \cdot \|\mathbf{y}\| - \frac{\partial}{\partial y_i}(\|\mathbf{y}\|) \cdot \mathbf{x}^T \mathbf{y}}{\|\mathbf{y}\|^2} \\ &= \frac{\mathbf{x}_i \cdot \|\mathbf{y}\| - \bar{y}_i \cdot \mathbf{x}^T \mathbf{y}}{\|\mathbf{y}\|^2} = (\mathbf{x}_i - q(\mathbf{y}) \cdot \bar{y}_i) / \|\mathbf{y}\|, \end{aligned}$$

¹Chair of Computer Graphics and Visualization, Technical University of Munich, Bavaria, Germany.

that is $\nabla q(\mathbf{y})^T = (\mathbf{x} - q(\mathbf{y}) \cdot \bar{\mathbf{y}}) / \|\mathbf{y}\|$. By applying the chain rule, for $k > 0$ we obtain

$$\begin{aligned} \nabla S_{\mathbf{x}}(\mathbf{y})^T &= \frac{\partial}{\partial \mathbf{y}}(\mathbf{x}^T \mathbf{y} \cdot q(\mathbf{y})^k) \\ &= q(\mathbf{y})^k \cdot \mathbf{x} + \mathbf{x}^T \mathbf{y} \cdot k q(\mathbf{y})^{k-1} \cdot \nabla q(\mathbf{y})^T \\ &= q(\mathbf{y})^k \cdot \mathbf{x} + k q(\mathbf{y})^k \cdot (\mathbf{x} - q(\mathbf{y}) \cdot \bar{\mathbf{y}}) \\ &= (k+1) q(\mathbf{y})^k \left(\mathbf{x} - \frac{k}{k+1} \mathbf{x}^T \bar{\mathbf{y}} \cdot \bar{\mathbf{y}} \right) \\ &= (k+1) q(\mathbf{y})^k \left(\mathbf{x} - \frac{k}{k+1} \bar{\mathbf{y}} \cdot \bar{\mathbf{y}}^T \mathbf{x} \right) \\ &= (k+1) q(\mathbf{y})^k \left(I - \frac{k}{k+1} \bar{\mathbf{y}} \bar{\mathbf{y}}^T \right) \mathbf{x} \end{aligned}$$

We then set

$$Z_k(\mathbf{y}) = Z_k(\mathbf{y})^T := Id - k / (k+1) \bar{\mathbf{y}} \bar{\mathbf{y}}^T$$

(as in the paper) to obtain

$$\nabla S_{\mathbf{x}}(\mathbf{y})^T = (k+1) q(\mathbf{y})^k Z_k(\mathbf{y}) \mathbf{x}.$$

Again by the chain rule, it follows that

$$\begin{aligned} \nabla(f_{\mathbf{x}} \circ P)(v_I) &= \nabla(S_{\mathbf{x}} \circ \tilde{N})(v_I) \\ &= \nabla S_{\mathbf{x}}(\tilde{N}(v_I)) D\tilde{N}(v_I) \\ &= (k+1) q(\tilde{N}(v_I))^k \mathbf{x}^T Z_k(\tilde{N}(v_I)) D\tilde{N}(v_I) \end{aligned}$$

Denoting $\tilde{N}(v_I)$ by \mathbf{y}_I and the Jacobian matrix $D\tilde{N}(v_I)$ by $D\mathbf{y}_I$, this expression simplifies to

$$\nabla(f_{\mathbf{x}} \circ P)(v_I) = (k+1) q(\mathbf{y}_I)^k \mathbf{x}^T Z_k(\mathbf{y}_I) D\mathbf{y}_I.$$

For $k = 0$, it holds that $\nabla(f_{\mathbf{x}} \circ P)(v_I) = \mathbf{x}^T D\mathbf{y}_I$. Because Z_0 evaluates to the identity matrix, independently of \mathbf{y}_I , the reformulation

$$\arg \min_{\mathbf{x} \in \Omega} \|\mathbf{x}^T Z_k(\mathbf{y}_I) D\mathbf{y}_I\|^2 \quad \text{s.t.} \quad Z_k(\mathbf{y}_I) \mathbf{x} \in C \quad (3)$$

is valid for $k = 0$ as well.

For $k \rightarrow \infty$, the matrix $Z_k(\mathbf{y}_I)$ converges to a projection matrix that introduces \mathbf{y}_I in the co-kernel of $Z_k(\mathbf{y}_I) D\mathbf{y}_I$. Hence, the solution to Opt. (3) converges to the normalized feature response $\bar{\mathbf{y}}_I$. This behavior is consistent with the observation that the significance measure $S_{\mathbf{x}}$ pushes the feature response closer to the feature objective \mathbf{x} when increasing k .

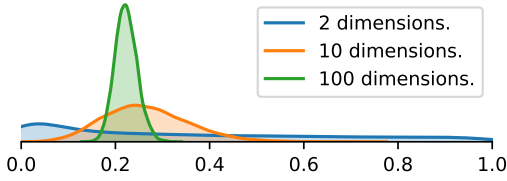


Figure S1. Distribution of sparsity values (obtained via the Hoyer measure (Hoyer, 2004)) for 10,000 uniformly sampled vectors from a sphere in 2/10/100-dimensional ambient space. A value of zero indicates no sparseness, i.e. all components are equal, whereas a value of one indicates a canonical vector.

Incorporating the Linear Constraint

Opt. (3) can be further simplified by integrating the linear constraint $Z_k(\mathbf{y}_I)\mathbf{x} \in C$ into the objective function via substitution. Let $C_M \in \mathbb{R}^{n_f \times \dim C}$ be a matrix such that its columns form an orthonormal basis of C . Then, \mathbf{x} satisfies the constraint in Opt. (3) iff $\mathbf{x} \in \text{Im } Z_k(\mathbf{y})^{-1}C_M$. Note that $Z_k(\mathbf{y})^{-1} = \text{Id} + k \cdot \overline{\mathbf{y}\mathbf{y}^T}$ always exists by the Sherman-Morrison formula. Since $Z_k(\mathbf{y})^{-1}C_M$ has full column-rank, the singular value decomposition (SVD) $U\Sigma V^T = Z_k(\mathbf{y})^{-1}C_M$ yields an orthogonal matrix U with $\text{Im } U = \text{Im } Z_k(\mathbf{y})^{-1}C_M$. The substitution of \mathbf{x} then is given by $\mathbf{x} = U\sigma$, for $\sigma \in \mathbb{R}^{\dim C}$ with $\|\sigma\| = 1$. Opt. (3) reduces to

$$\arg \min_{\|\sigma\|=1} \|\sigma^T U^T Z_k(\mathbf{y}) D \mathbf{y}\|^2.$$

S2. Sampling Strategy for Target Objectives

When selecting the target objectives, we noticed that uniformly sampling the nD -sphere Ω does not yield representative target objectives. Due to the curse of dimensionality, the density of sparse target objectives, especially canonical objectives e_i representing single features, rapidly tends to 0 even for medium sizes of n (see Fig. S1). However, sparse target objectives are usually those which are investigated in FV. Hence, we implement a sampling strategy as follows: First, a scalar value s in $[0, 1]$ is sampled from a uniform distribution. It indicates how sparse the target objective \mathbf{x} is supposed to be. We use the Hoyer sparseness measure (Hoyer, 2004) to measure sparseness of \mathbf{x} . It measures the ratio of $\|\mathbf{x}\|_1$ and $\|\mathbf{x}\|_2$, and then applies an affine transformation such that the smallest possible ratio of 1:1 (for canonical vectors) maps to 1 and the largest possible ratio of \sqrt{n} :1 (for vectors with all components being equal) maps to 0. Second, we (uniformly) sample a random vector \mathbf{x}_0 from Ω and run curvilinear search (Wen & Yin, 2013) with \mathbf{x}_0 as initial value to find a vector $\mathbf{x} \in \Omega$ that minimizes $(\text{hoyer}(\mathbf{x}) - s)^2$.

When experimenting with an optimization based approach of finding critical spaces (which we dropped due to the lack of reliability), we observed that introducing sampling as above highly increases the chance of finding the global optimum. If target objectives are sampled uniformly from Ω , the optimization process commonly gets stuck in a slightly worse than globally optimal solution C^* that does not generalize well to sparse target objectives. That is, the solution of Opt. (3) w.r.t. the critical space C^* matches the projected target objective $\tilde{\mathbf{x}}$ when providing a realization of a sampled target objective, but does not so when providing a realization to a canonical target objective e_i . Since the proposed sampling strategy helps in this regard, we utilize it in our experiments as well to avoid potentially skewed results.

S3. Realizations and Difference maps

In our work, we commonly compare angular distances to SSIM indices. Since SSIM is just one of many potential measures to quantify similarity between images, we additionally provide difference maps in Fig. S2 and argue that SSIM represents these accurately. The shown excerpt of GoogLeNet-inception4c samples reveals that predicted objectives mostly behave similarly to target objectives in re-optimization, whereas feature responses introduce significant high frequency changes of the same magnitude as randomly picked feature vectors. Note that the outliers produced by our method also show up in the difference maps. One is contained in the excerpt. It corresponds to a blue outlier point of Fig. 2a in the paper that is located next to the red cluster. We do not provide difference maps for all experiments to keep the size of the supplementary file to reasonable sizes. We refer to the experiment suite provided in the source code submission for creating further imagery.

References

- Hoyer, P. O. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5 (Nov):1457–1469, 2004.
- Wen, Z. and Yin, W. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1):397–434, Dec 2013. ISSN 1436-4646. doi:10.1007/s10107-012-0584-1.

S4. Detailed Results of Experiments

In the “Results and Evaluation” section of our manuscript we perform a number of experiments and summarize the resulting findings. To further back up these findings, we provide several additional statistics below. For each experiment, we show the following diagrams:

1. A stacked bar chart depicting mean contributions α_j of j -th left-singular vectors—ordered by singular values—to target objectives (details see paper).
2. A kernel density estimation (kde) plot showing the distribution of SSIM indices when re-optimizing samples w.r.t. the target objectives. A sample is considered unstable if its SSIM index drops below 0.7 (red, dashed line).
3. A bivariate plot relating angular distances to target objectives and SSIM indices when re-optimizing samples w.r.t. the predicted objective. Triangular markers correspond to unstable samples. If present, point size encodes the fraction of the target objective that lives in the respective critical space.

The following experiments are conducted:

ID	Network	Features	Aggregation	Cos. term	Param.
The Simple Case					
1	GoogLeNet	512 filters, inception4c	Max pooling + mean aggr.	$k = 2$	RGB
Utilizing the Critical Space					
2	DenseNet	10 neurons, classification	–	$k = 2$	RGB
3	DenseNetEx4	10 neurons, classification	–	$k = 2$	RGB
4	DenseNet	10 neurons, classification	–	$k = 0$	RGB
5	DenseNetEx4	10 neurons, classification	–	$k = 0$	RGB
Dropping the Cosine Term					
6	DenseNet	32 to 342 filters, dense3	Max pooling + mean aggr.	$k = 0$	RGB
7	GoogLeNet	32 to 512 filters, inception4c	Max pooling + mean aggr.	$k = 0$	RGB
8	DenseNet	80 filters, dense3	Max pooling + mean aggr.	$k = 0$	RGB
9	DenseNet	80 filters, dense3	mean aggr.	$k = 0$	RGB
10	DenseNet	80 filters, dense3	pick center neuron	$k = 0$	RGB
11	DenseNet	32 to 342 filters, dense3	Max pooling + mean aggr.	$k = 2$	RGB
12	GoogLeNet	32 to 512 filters, inception4c	Max pooling + mean aggr.	$k = 2$	RGB
13	DenseNet	80 filters, dense3	Max pooling + mean aggr.	$k = 2$	RGB
14	DenseNet	80 filters, dense3	mean aggr.	$k = 2$	RGB
15	DenseNet	80 filters, dense3	pick center neuron	$k = 2$	RGB
Different Parameterizations					
16	DenseNet	342 filters, dense3	Max pooling + mean aggr.	$k = 2$	RGB
17	DenseNet	342 filters, dense3	Max pooling + mean aggr.	$k = 2$	FFT
18	DenseNet	342 filters, dense3	Max pooling + mean aggr.	$k = 2$	FFT-E
19	GoogLeNet	512 filters, inception4c	Max pooling + mean aggr.	$k = 2$	RGB
20	GoogLeNet	512 filters, inception4c	Max pooling + mean aggr.	$k = 2$	FFT
21	GoogLeNet	512 filters, inception4c	Max pooling + mean aggr.	$k = 2$	FFT-E
22	SRNet	64 filters, 8-th block	Max pooling + mean aggr.	$k = 2$	mask = 1
23	SRNet	64 filters, 8-th block	Max pooling + mean aggr.	$k = 2$	mask $\in [-1, 1]$
24	SRNet	64 filters, 8-th block	Max pooling + mean aggr.	$k = 0$	mask = 1
25	SRNet	64 filters, 8-th block	Max pooling + mean aggr.	$k = 0$	mask $\in [-1, 1]$

Inverting the Feature Visualization Process for Feedforward Neural Networks

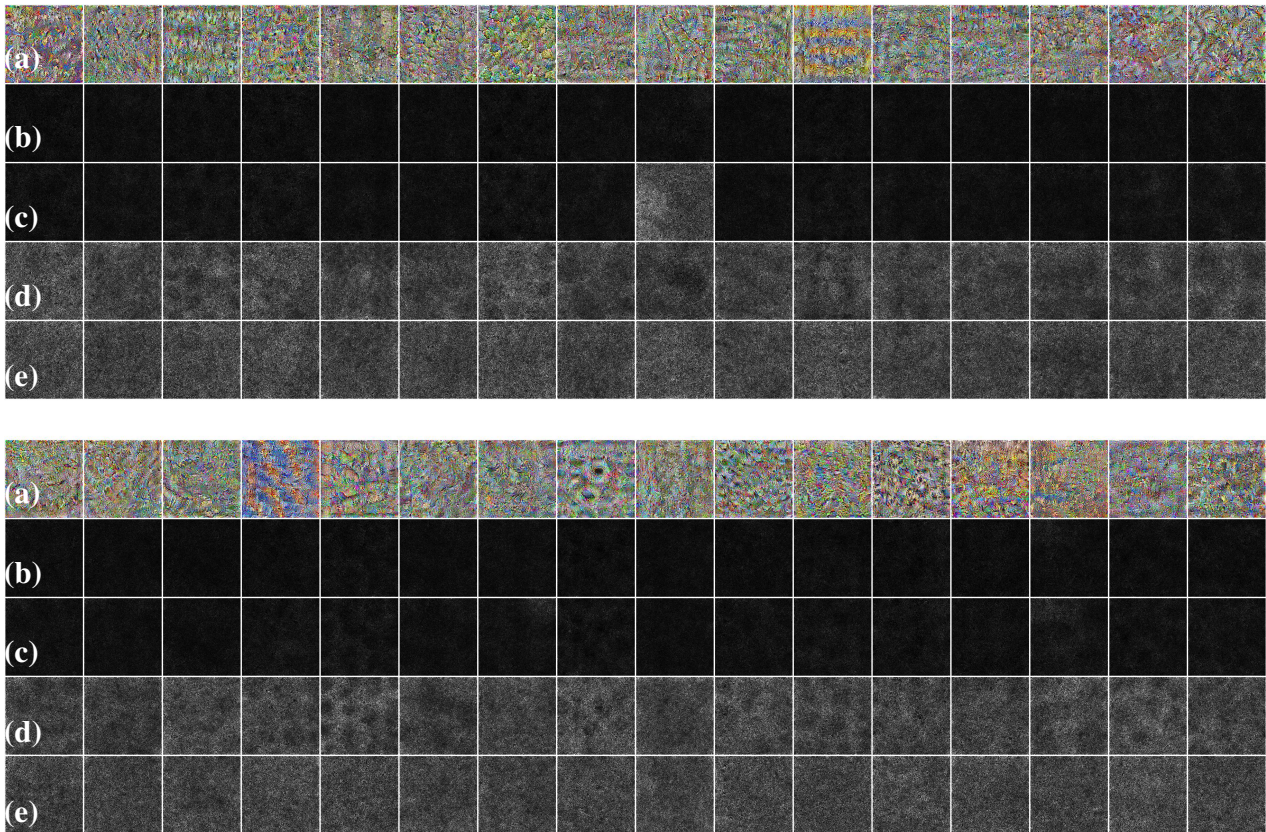
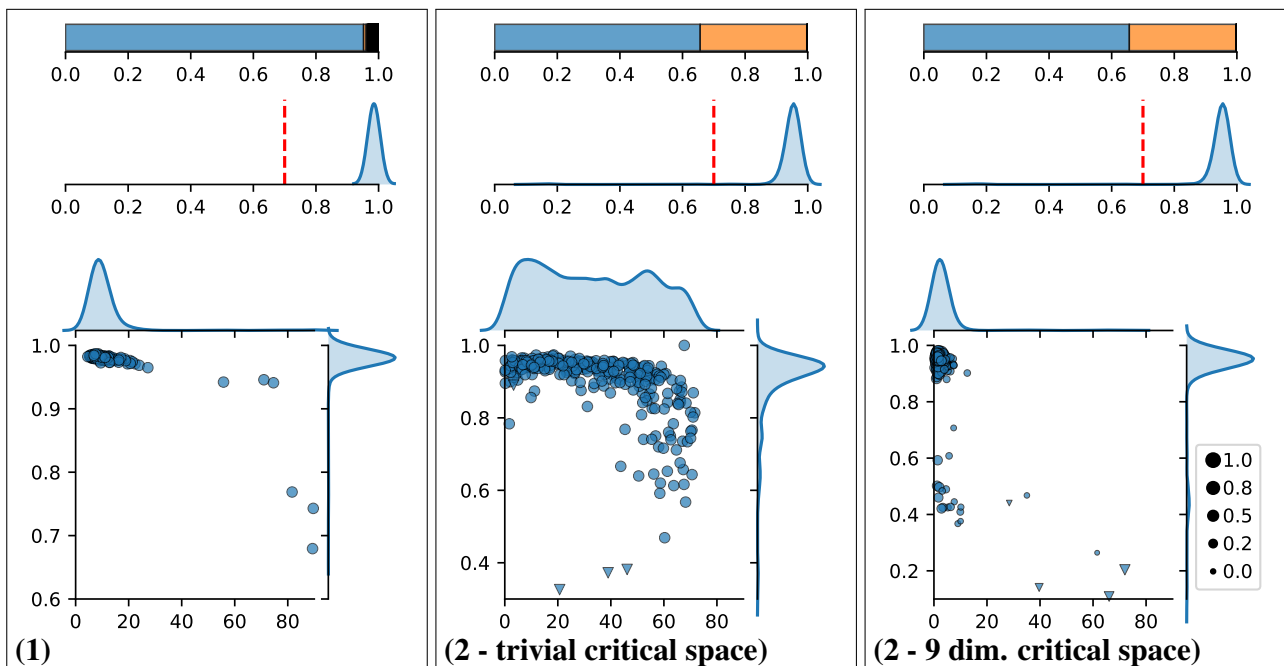
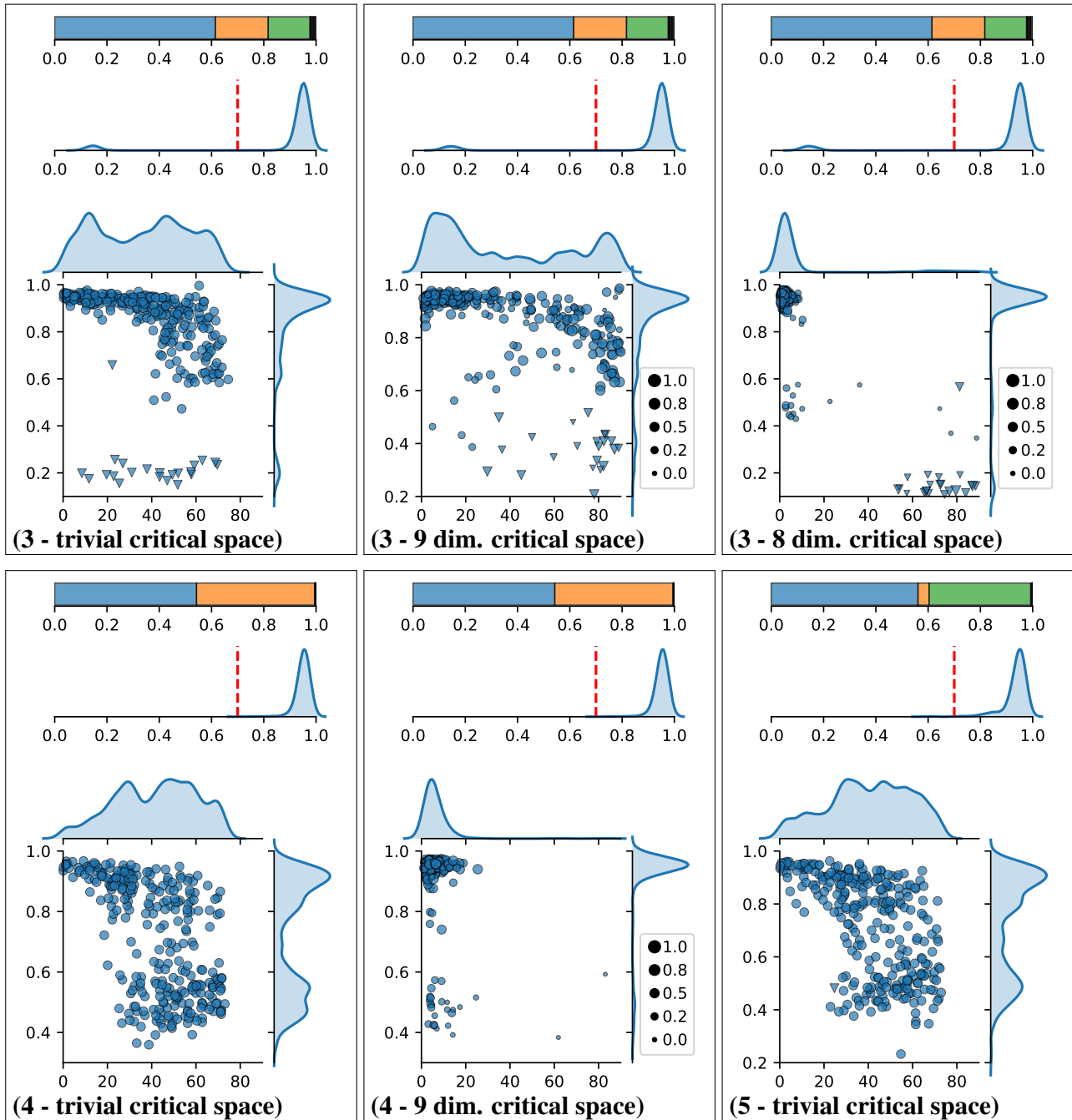


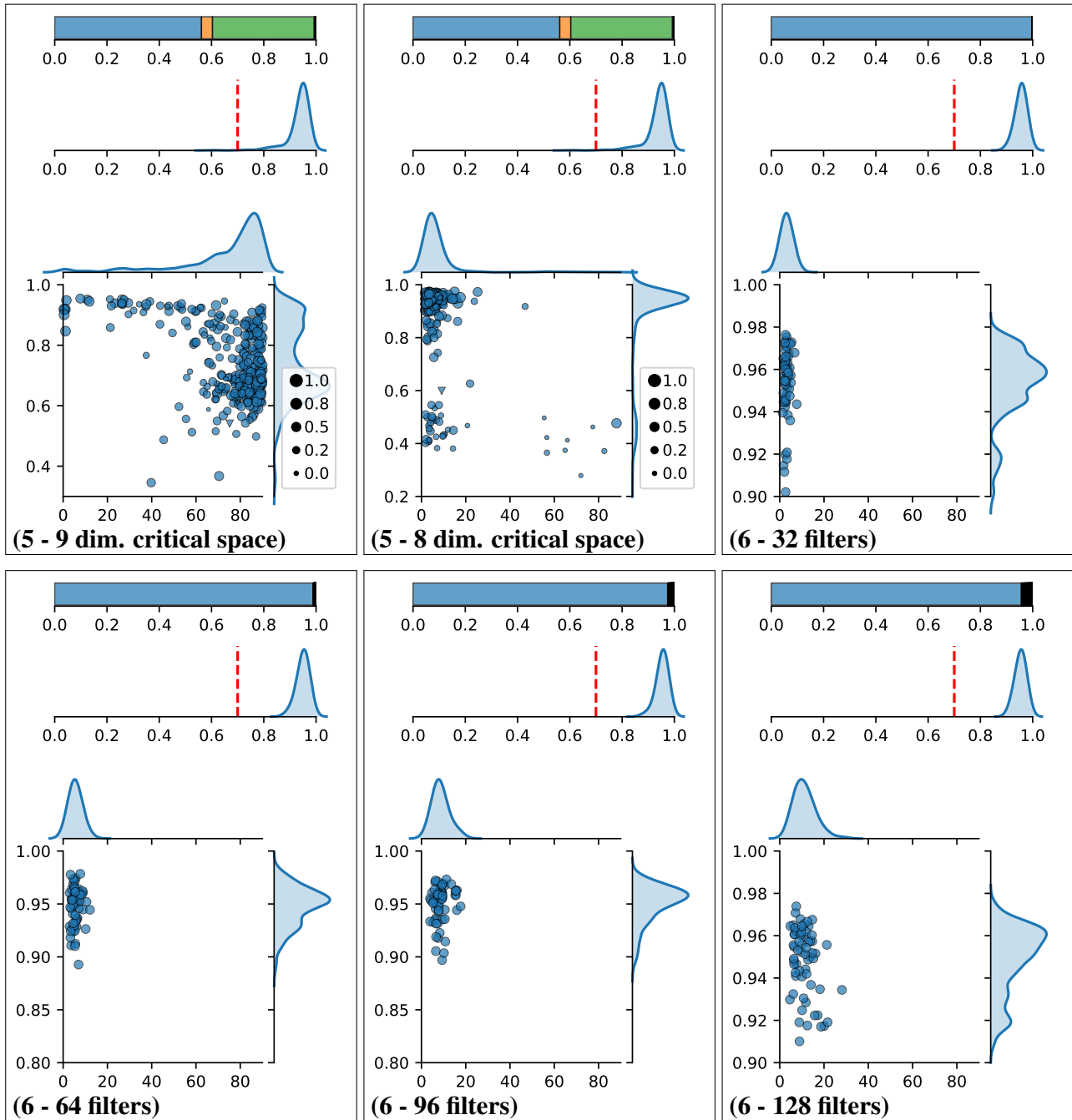
Figure S2. (a) Excerpt of realizations for experiment (1). Difference maps obtained when re-optimizing w.r.t. *(b)* the target objective, *(c)* the predicted objective, *(d)* the feature response or *(e)* a random feature vector. Differences are computed per pixel by taking the maximal difference over the three RGB channels. Best viewed electronically. Note that the image has been compressed to comply with arXiv file size constraints. Nonetheless, perceived noise and checkerboard artifacts occur in the uncompressed version as well and cannot be ascribed to image compression.



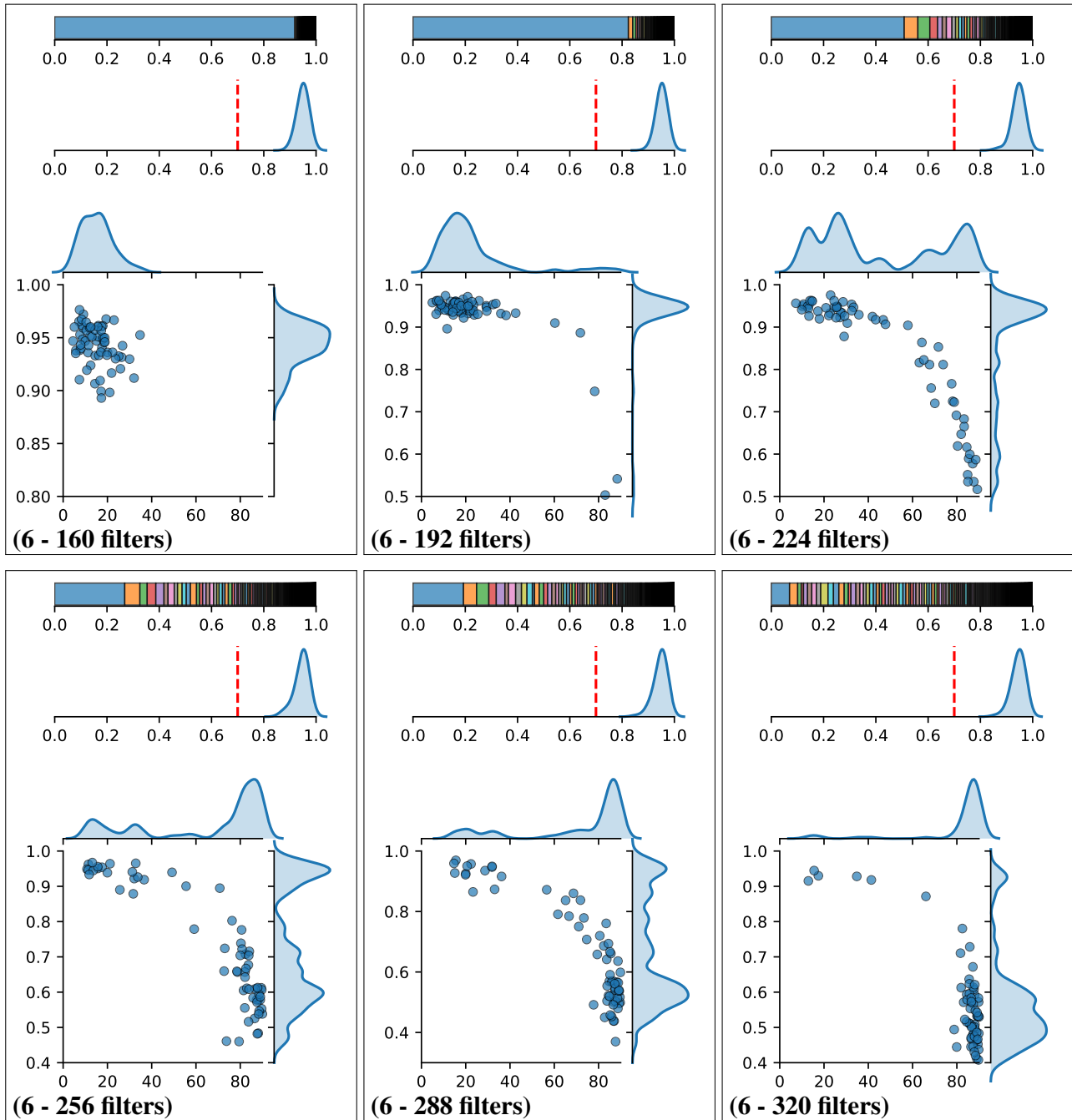
Inverting the Feature Visualization Process for Feedforward Neural Networks



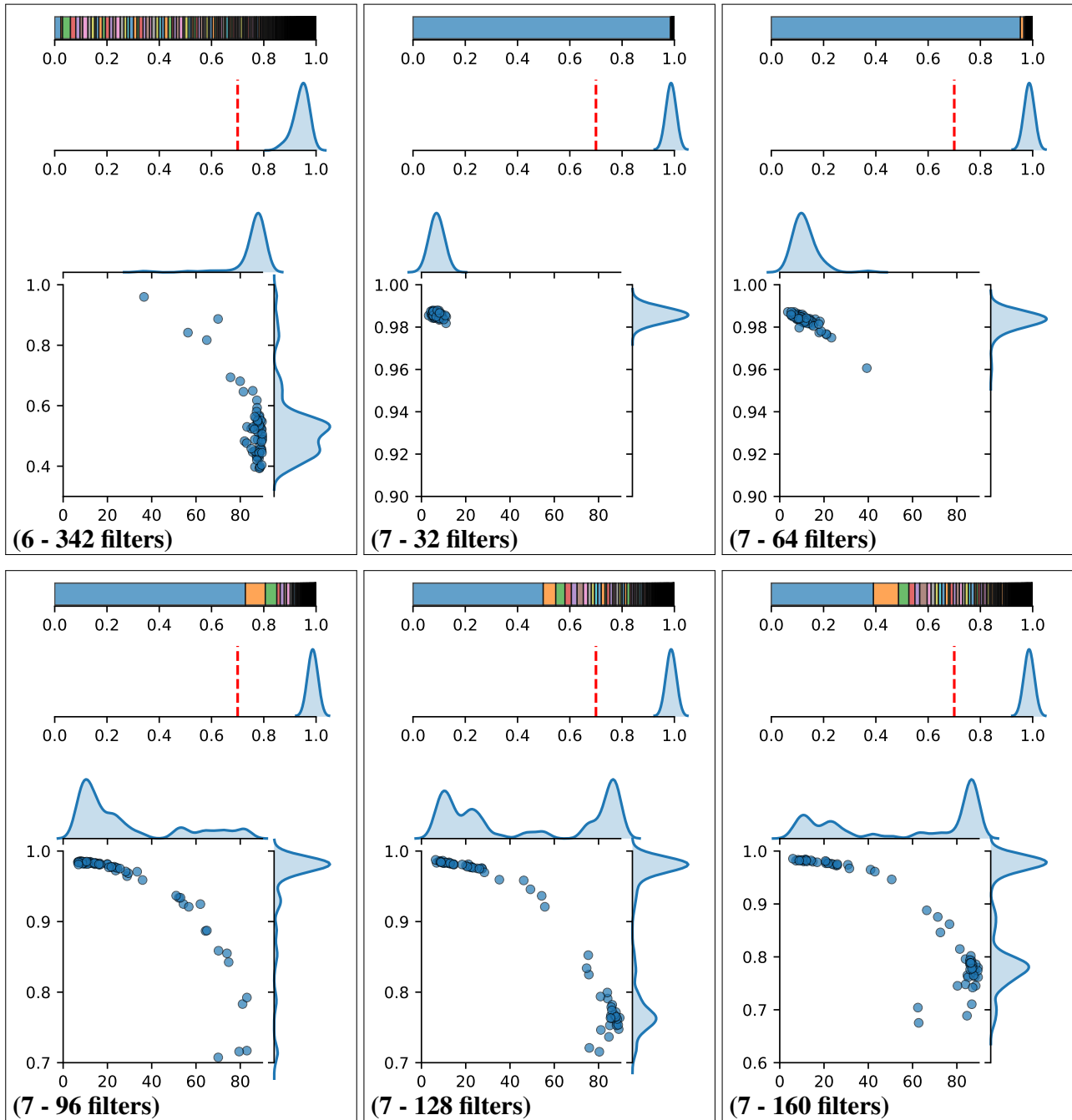
Inverting the Feature Visualization Process for Feedforward Neural Networks



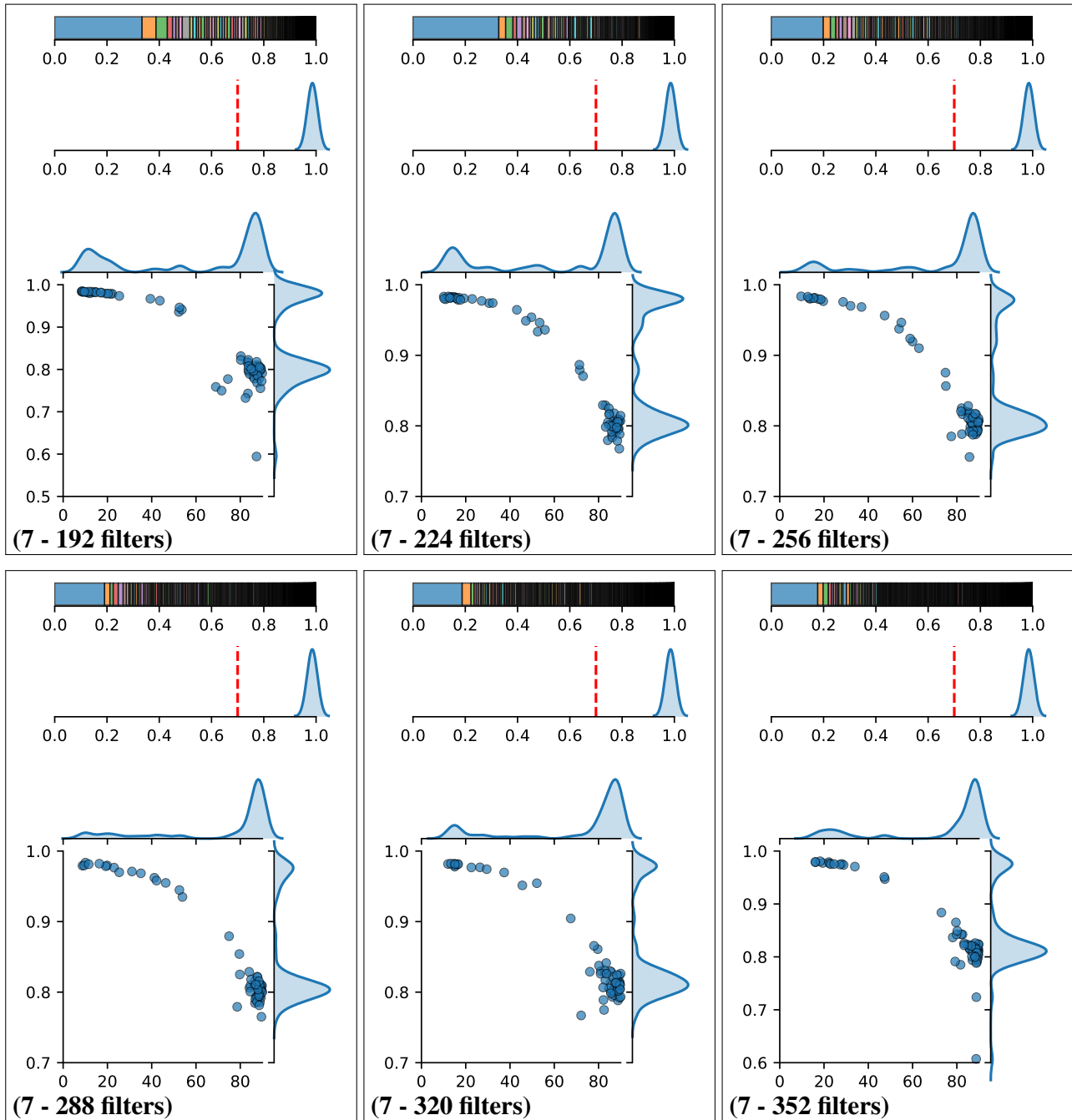
Inverting the Feature Visualization Process for Feedforward Neural Networks



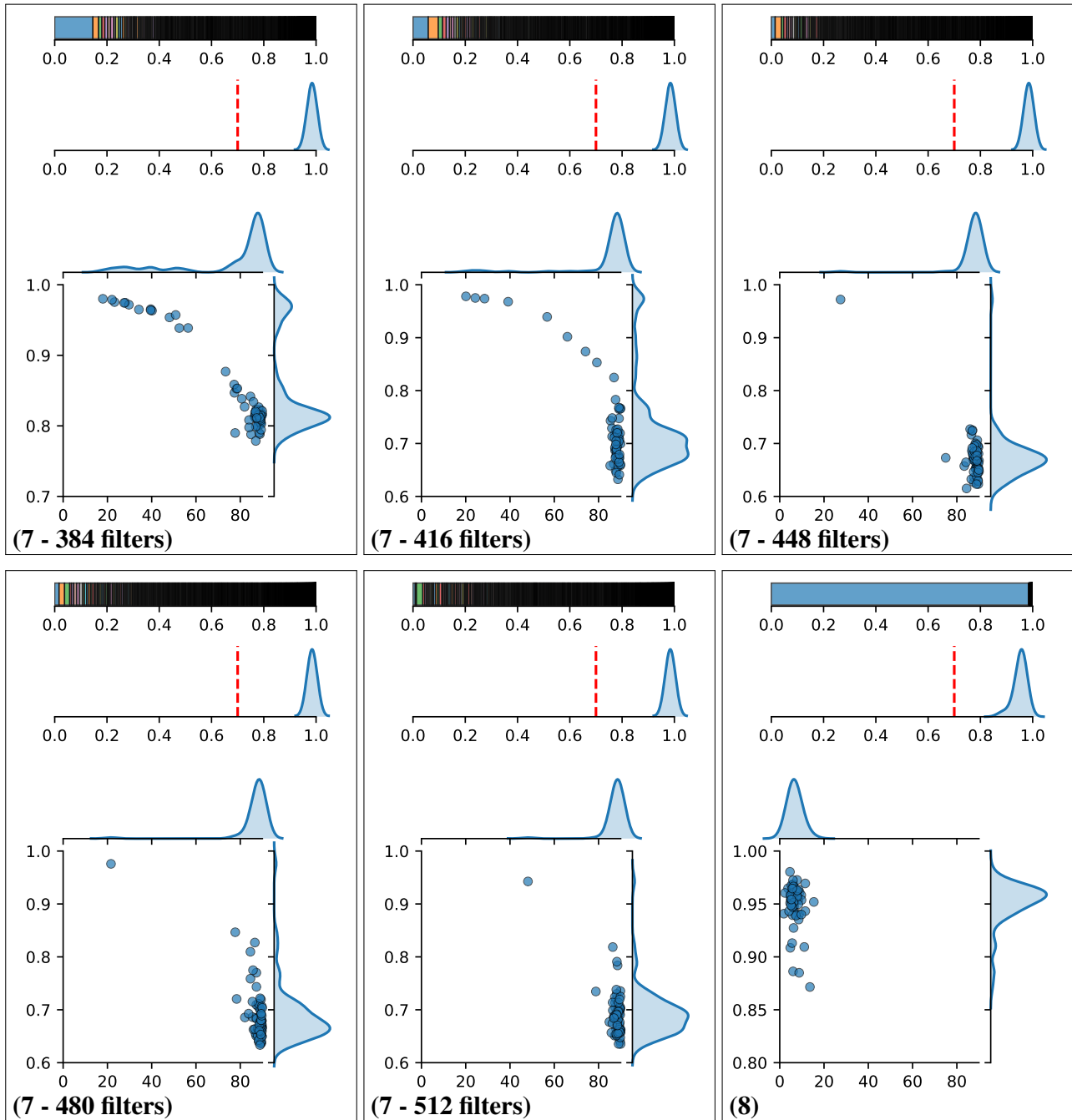
Inverting the Feature Visualization Process for Feedforward Neural Networks



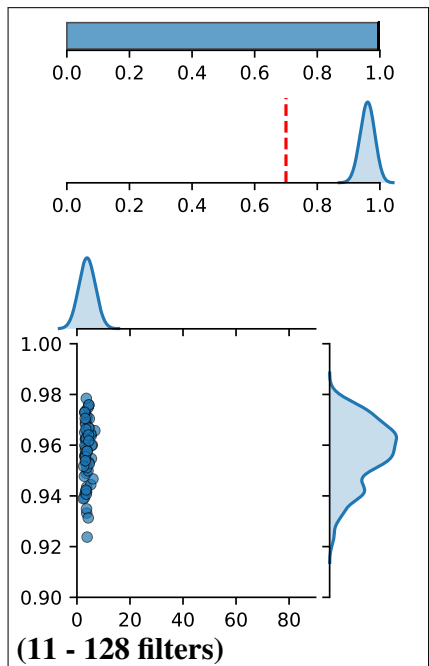
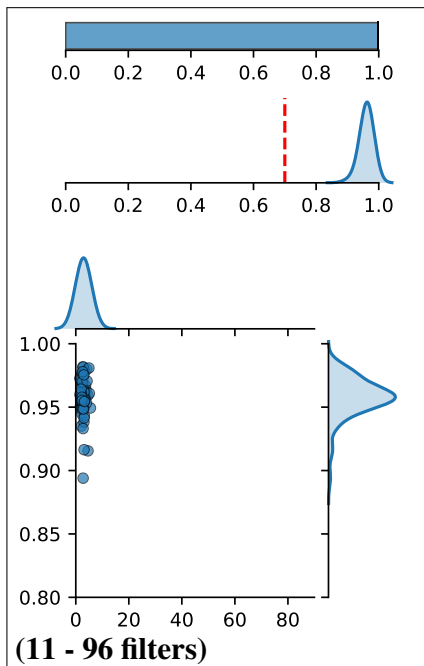
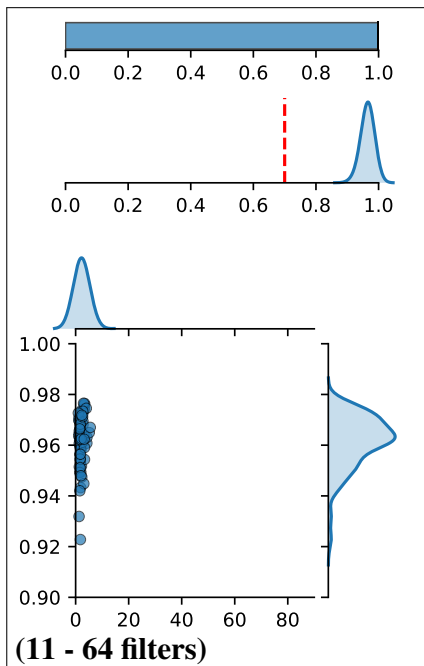
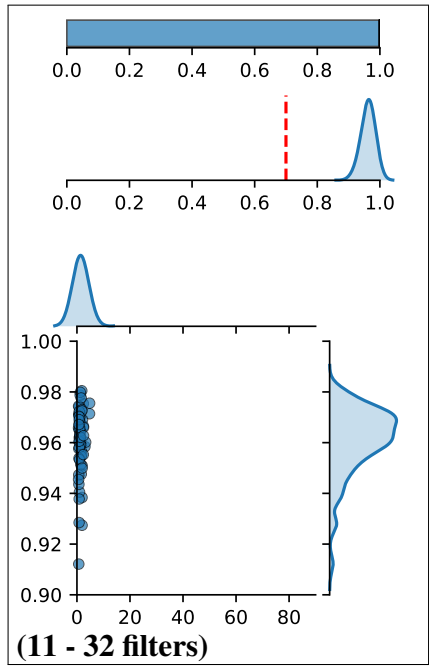
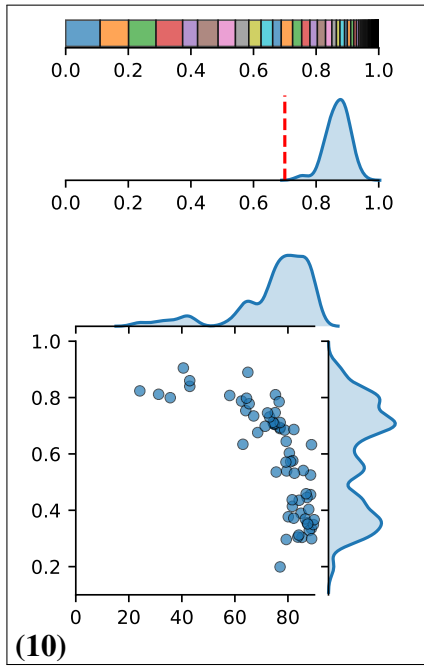
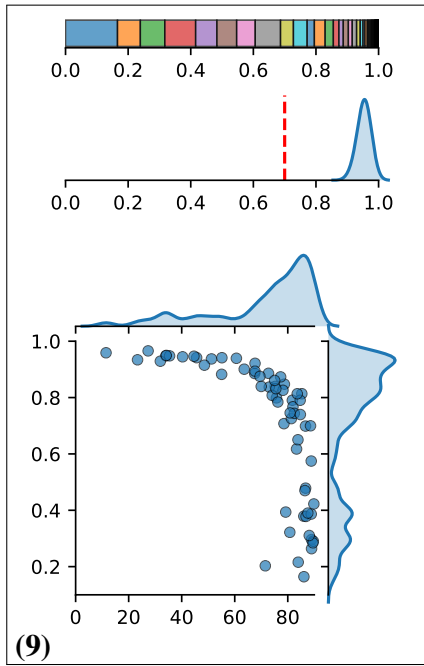
Inverting the Feature Visualization Process for Feedforward Neural Networks



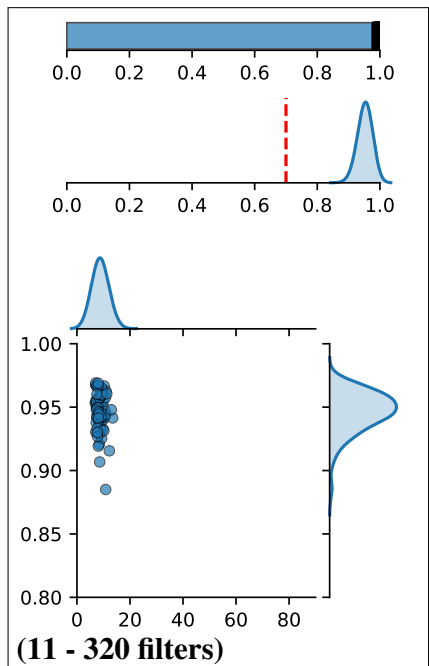
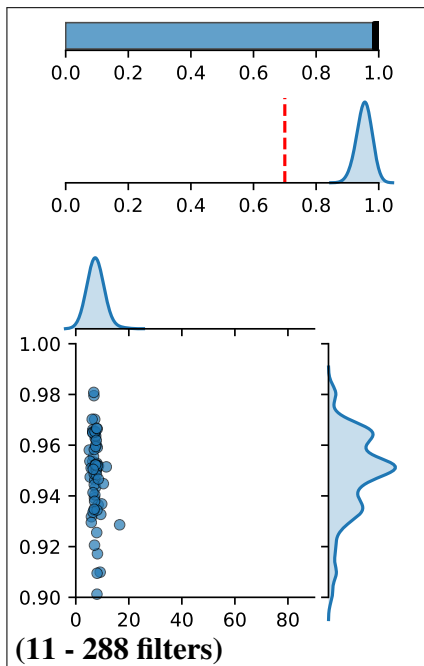
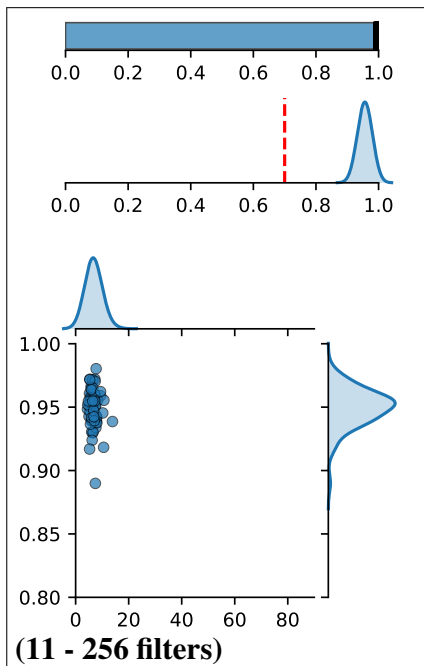
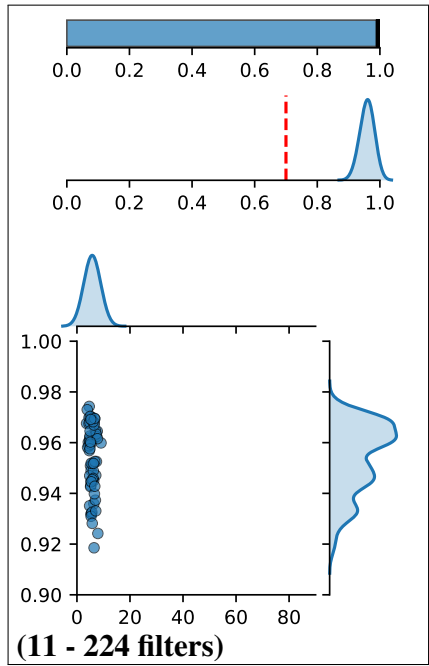
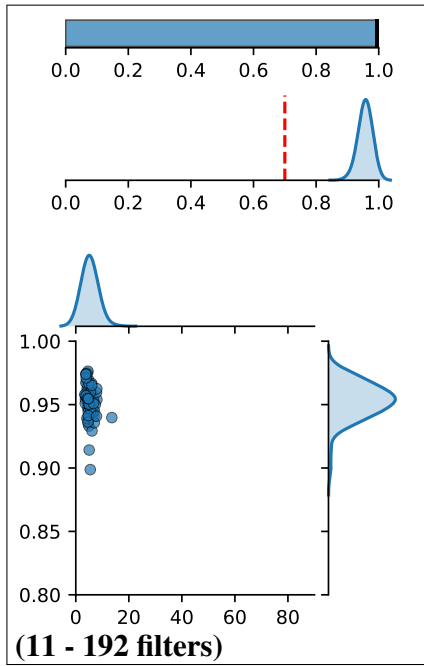
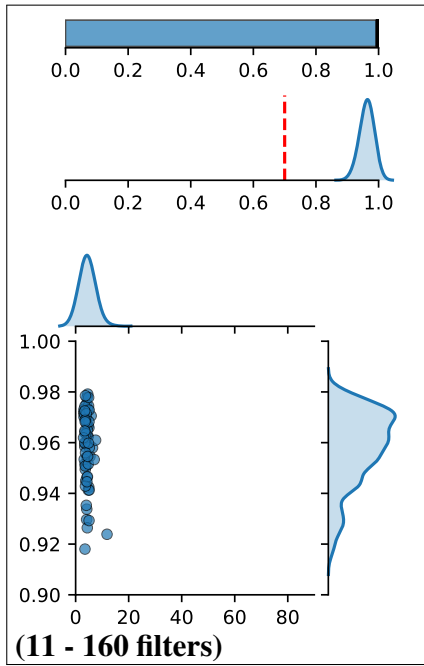
Inverting the Feature Visualization Process for Feedforward Neural Networks



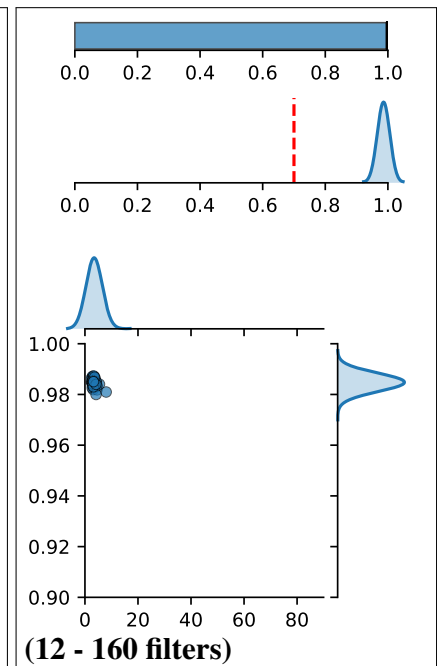
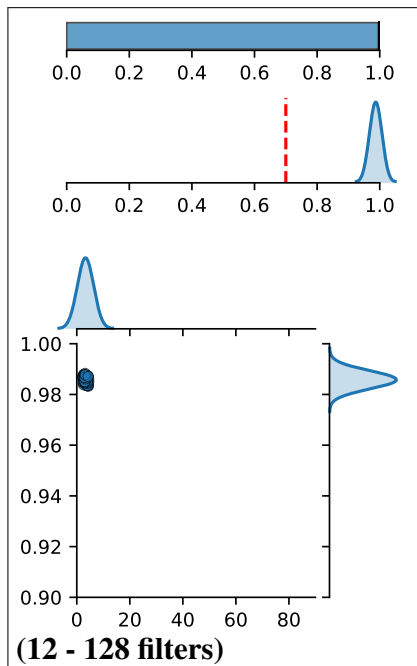
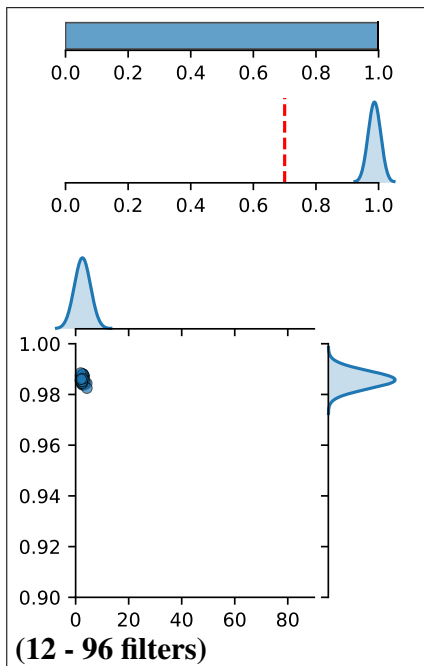
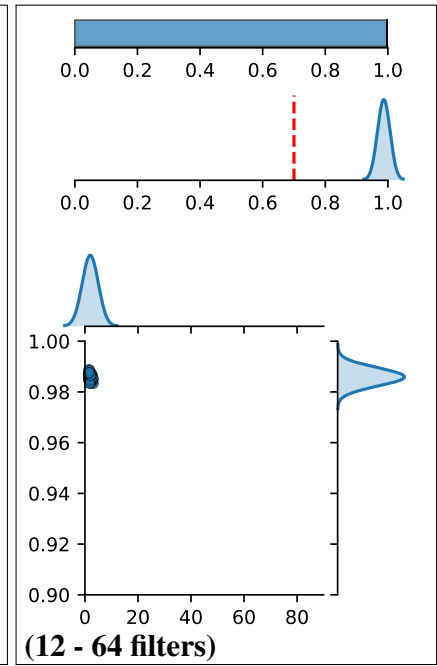
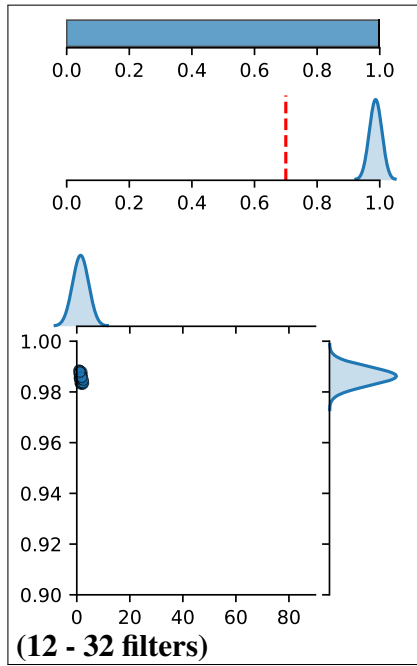
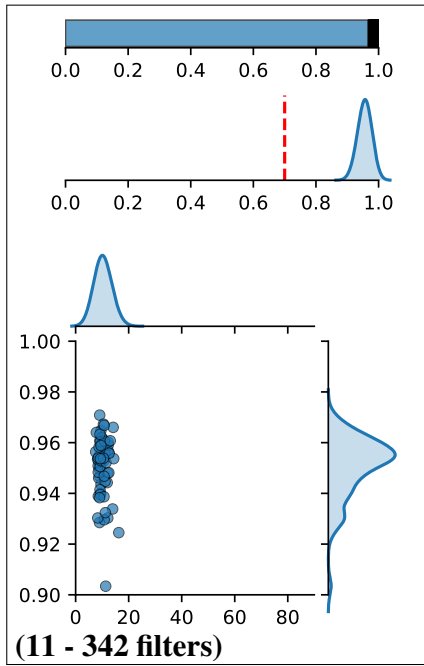
Inverting the Feature Visualization Process for Feedforward Neural Networks



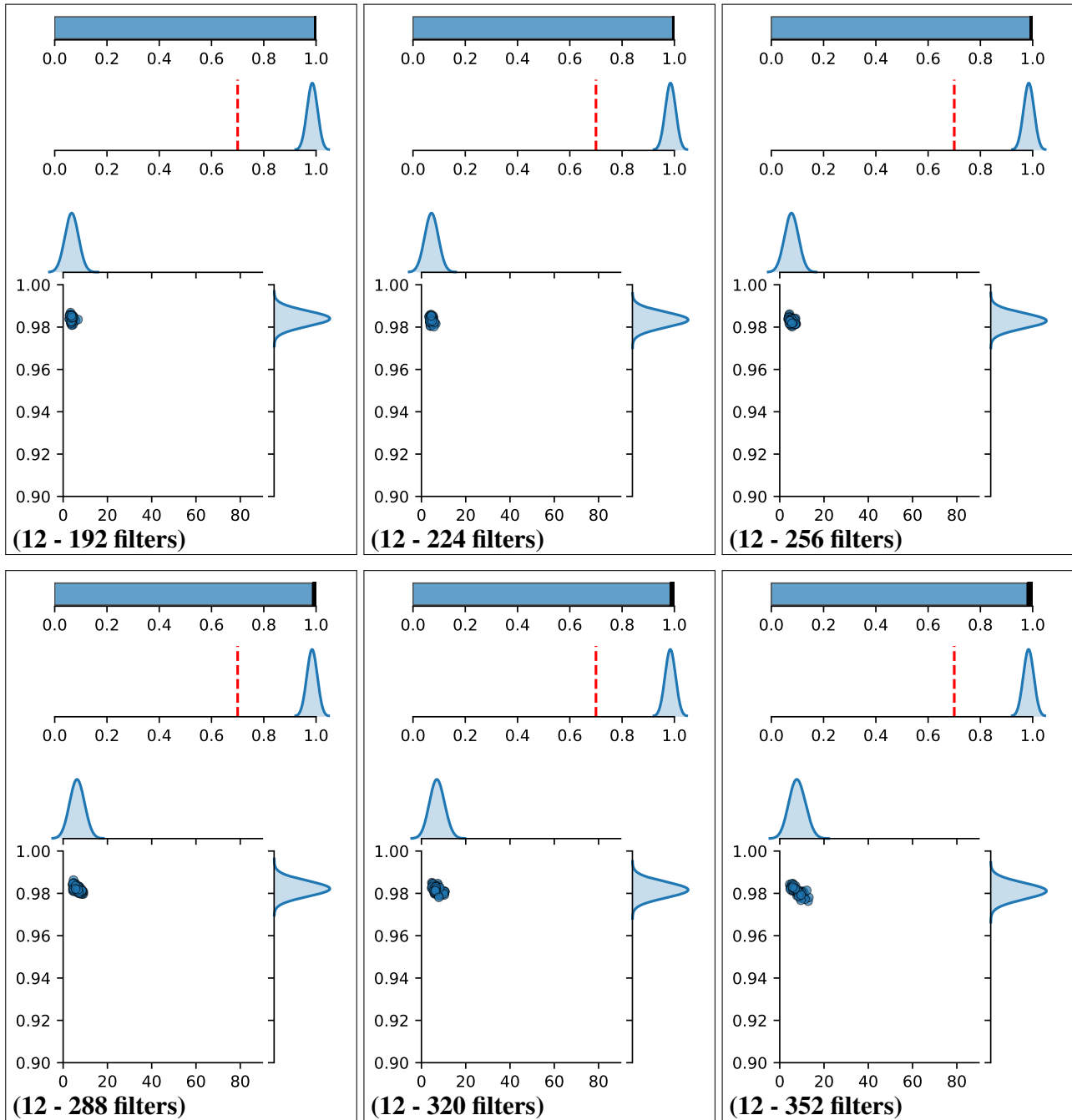
Inverting the Feature Visualization Process for Feedforward Neural Networks



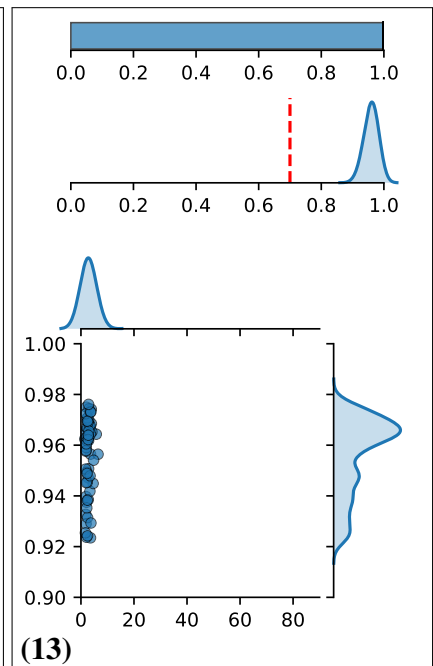
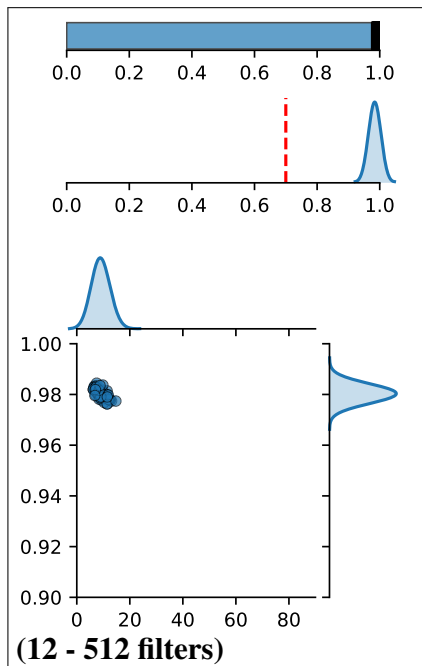
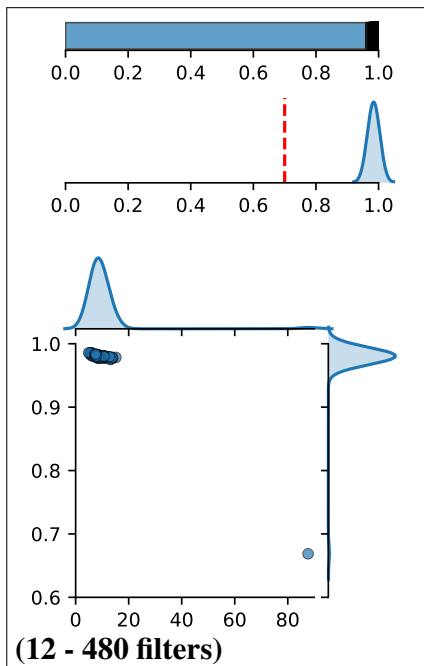
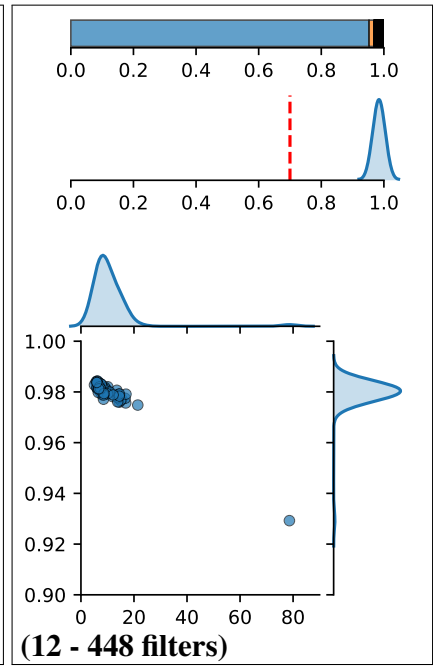
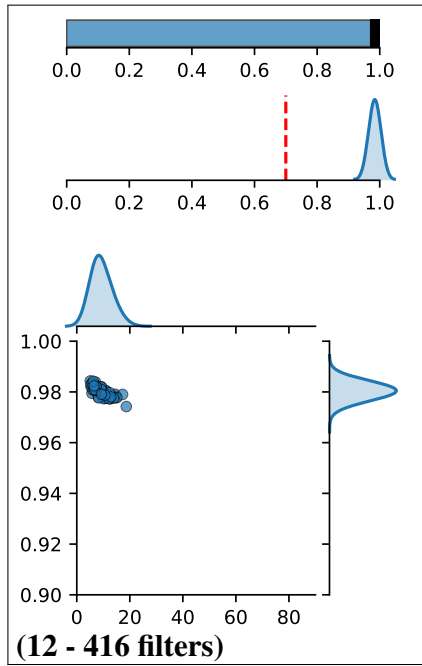
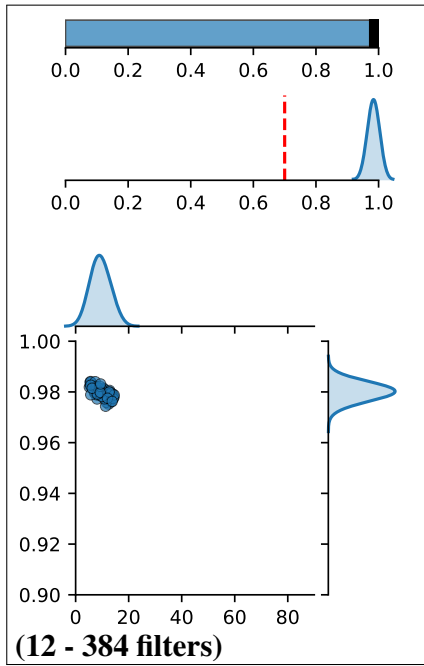
Inverting the Feature Visualization Process for Feedforward Neural Networks



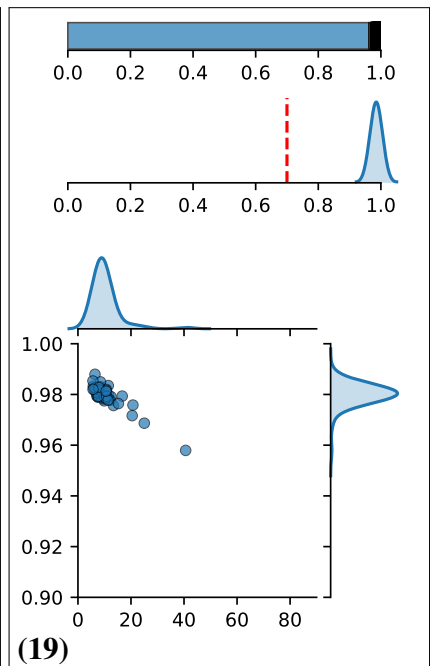
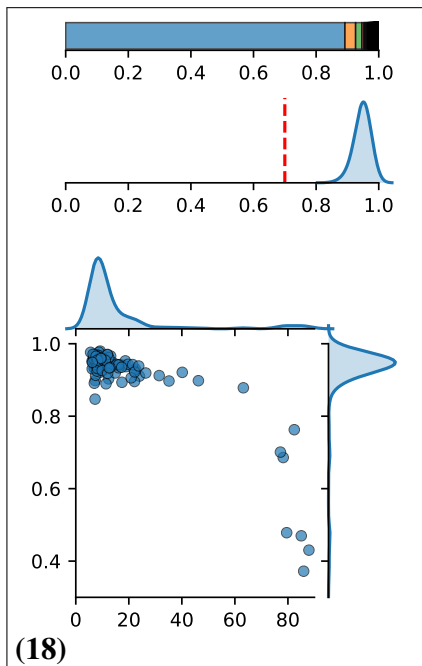
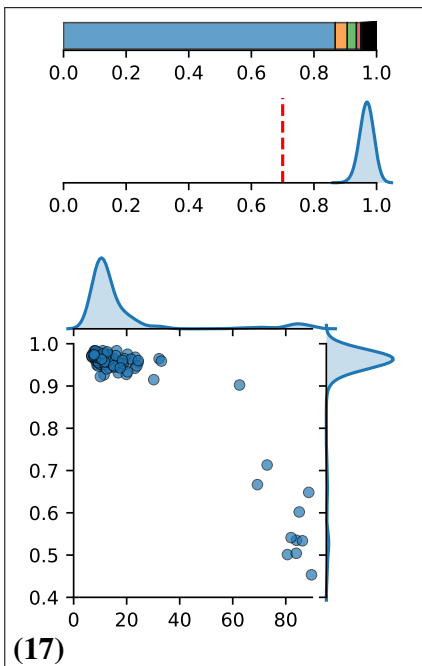
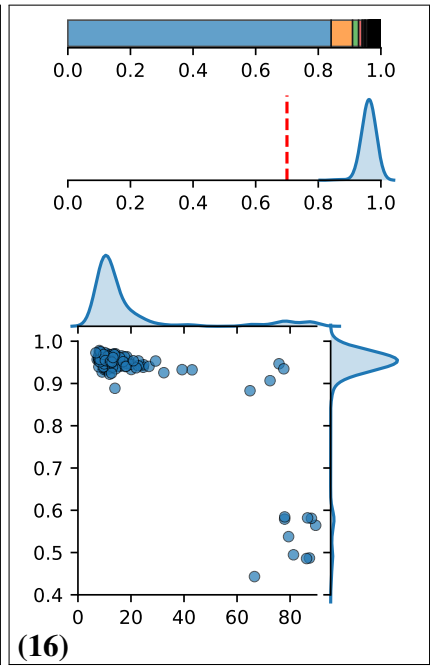
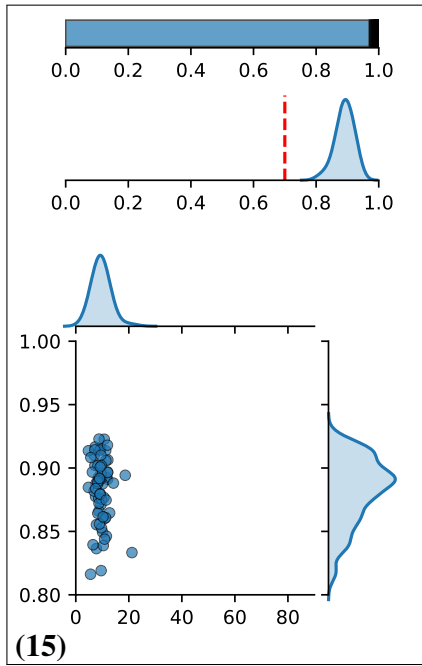
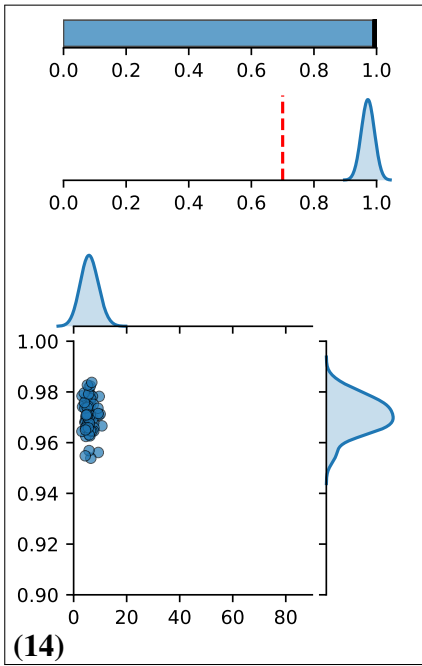
Inverting the Feature Visualization Process for Feedforward Neural Networks



Inverting the Feature Visualization Process for Feedforward Neural Networks



Inverting the Feature Visualization Process for Feedforward Neural Networks



Inverting the Feature Visualization Process for Feedforward Neural Networks

