TUM School of Computation, Information and Technology
Technische Universität München

Temporally Coherent Video Generation with Generative Adversarial Networks

You Xie

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.

Vorsitz:

Prof. Dr. Gudrun J. Klinker

Prüfer*innen der Dissertation:

1. Prof. Dr.-Ing. Nils Thuerey
2. Assistant Prof. Dr. Angela Yao,
   National University of Singapore

Die Dissertation wurde am 12.07.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 11.12.2022 angenommen.

0 Acknowledgement

I would like to express my heartfelt gratitude to everyone who has helped and accompanied me during my doctoral studies. The journey is full of ups and downs, but their help makes me hold on to the end. First of all, I would like to express my heartfelt thanks to my supervisor Nils Thuerey. He guided me into the field of physical simulation, led me to apply deep learning approaches to physical simulation, and patiently taught me different research skills in different situations. He can always give me constructive suggestions to help me out of the woods. Besides, he influenced me by example and taught me how to deal with the various problems encountered in the research process through his own actions. His support encourages me not to give up and to persevere to the end.

Furthermore, I am very grateful for the opportunities to work with Laura Leal-Taix*é* (from TUM) and Angela Yao (from NUS), whose suggestions are always enlightening and helpful. It is my honor to have Angela Yao and Gudrun Klinker as my thesis committee. At the same time, I also want to express my gratitude to the co-authors of our projects, Mengyu Chu, Eric Franz, Maximilian Werhahn, Jonas Mayer, Huiqi Mao, Ziwei Yu, Lingling Yang, Ping Chen. Working with them is pretty efficient and pleasant and we can always achieve tacit agreements at work. Besides, it has always been a great pleasure to meet Barbara Solenthaler (from ETH Zurich), Chris Wojtan (from IST Austria), and their groups via our yearly retreat trips. It is really wonderful to introduce each other's projects and conduct interesting discussions to share their views.

Moreover, I would like to thank all my colleagues, and their company let me never feel lonely over these years. Special thanks to Kiwon Um and Mengyu Chu, who provide me with a lot of help during my research, to Marie-Lena Eckert, Kern Michael for organizing interesting group activities; to Liwei Chen, Erik Franz, Georg Kohl, Brener d'L*é*lis, Bjoern List, Philipp Holl, Lukas Prantl, Patrick Schnell, Nilam Tathawadekar, Benjamin Holzschuh, Robin Greif, Shuvayan Brahmachary, Rene Winchenbach, Wei He, Stephan Rap, Steffen Wiewel for helpful discussion on research topics and others as well; to R*ü*diger Westermann, Matthias Niessner, Angela Dai, Ji Hou, and Justus Thies for interesting discussions during lunch. My thanks extend to my co-workers, Andreas R*ö*ssler, Dejan Azinovic, Zhenyu Chen, Manuel Dahnert, Kevin H*ö*hlein, Christian Reinbold, Junpeng Wang, and Ludwic Leonard Mendez for their accompany; to Susanne

Weitz and Sebastian Wohner for generous help. Because of you, my journey at TUM is really meaningful and unforgettable.

Last but not least, I would like to thank my friends, my parents, and my whole family, for their understanding and concern. Their support is a source of motivation for me to persevere.

0 Abstract

Video generation has been an important topic in both computer vision and graphics, but still challenging because of the complicated dynamics in the sequence. In this dissertation, we will explore GAN-based video generation algorithms under various backgrounds, such as fluid flow and natural videos. The primary goal of our work is to apply GANs to achieve realistic and temporally coherent details in the generated sequences.

Increasing the resolution of a fluid simulation always always brings tremendous computation costs. We propose a temporally coherent generative model for fluid flow super-resolution. Based on a conditional GAN, our model generates consistent and detailed results via a novel temporal discriminator, in addition to the commonly used spatial one. Our experiments show that the generator is able to infer more realistic high-resolution details by using additional physical quantities, such as the low-resolution velocities or vorticities, which offer means for artistic control as well. We additionally employ a physics-aware data augmentation step, which is crucial to avoid overfitting and reduce the memory requirements. In this way, our network learns to generate advected quantities with highly detailed, realistic, and temporally coherent features. Our method works instantaneously, using only a single time-step of low-resolution fluid data. To improve the efficiency of model training with 3D volume data, we re-visit the classic idea of unsupervised autoencoder pretraining and propose a modified variant that relies on a full reverse pass trained in conjunction with a given training task. This yields networks that are *as-invertible-as-possible*, and share mutual information across all constrained layers. We additionally establish links between singular value decomposition and pretraining and show how it can be leveraged for gaining insights into the learned structures. Most importantly, we demonstrate that our approach yields an improved performance for a wide variety of relevant learning and transfer tasks ranging from fully connected networks over residual neural networks to GANs.

For natural videos, we focus on pose-guided human video generation. Instead of generating the human sequence directly, we propose a novel approach to generate temporally coherent UV coordinates. Our method is not constrained by human body outlines and can capture loose garments and hair. We implemented a differentiable pipeline to learn UV mapping between a sequence of RGB inputs and textures via UV coordinates. Instead of treating the UV coordinates of each frame separately, our data generation

approach connects all UV coordinates via feature matching for temporal stability. Subsequently, a generative model is trained to balance the spatial quality and temporal stability. It is driven by supervised and unsupervised losses in both UV and image spaces. Our experiments show that the trained models output high-quality UV coordinates and generalize to new poses. Once a sequence of UV coordinates has been inferred by our model, it can be used to flexibly synthesize new looks and modified visual styles. Compared to existing methods, our approach reduces the computational workload to animate new outfits by several orders of magnitude.

Our methods achieved spatially realistic and temporally coherent sequence generation with GANs. Our various results, e.g., generated fluid flow and natural sequence, demonstrate the capability of GANs in video generation. We hope that our work can provide inspiration for other video generation tasks.

0 Zusammenfassung

Die Videogenerierung ist aufgrund der komplizierten Dynamik in der Sequenz ein wichtiges, aber immer noch schwieriges Thema in den Bereichen Computer Vision und Grafik. In dieser Dissertation erforschen wir GAN-basierte Algorithmen zur Videogenerierung vor verschiedenen Hintergründen, wie z. B. Flüssigkeitsströmungen und natürliche Videos. Das Hauptziel unserer Arbeit ist die Anwendung von GANs, um realistische und zeitlich kohärente Details in den erzeugten Sequenzen zu erreichen.

Bei Strömungssimulationen ist die Erhöhung der Auflösung immer mit enormen Rechenkosten verbunden. Wir schlagen ein zeitlich kohärentes generatives Modell für die Superauflösung von Strömungen vor. Basierend auf einem bedingten GAN erzeugt unser Modell konsistente und detaillierte Ergebnisse durch die Verwendung eines neuartigen zeitlichen Diskriminators, zusätzlich zu dem üblicherweise verwendeten räumlichen Diskriminator. Unsere Experimente zeigen, dass der Generator in der Lage ist, realistischere hochaufgelöste Details abzuleiten, indem er zusätzliche physikalische Größen verwendet, wie z. B. niedrig aufgelöste Geschwindigkeiten oder Wirbelstärken, die ebenfalls Mittel zur künstlerischen Kontrolle bieten. Zusätzlich verwenden wir einen physikalischen Datenanreicherungsschritt, der entscheidend ist, um eine Überanpassung zu vermeiden und den Speicherbedarf zu reduzieren. Auf diese Weise lernt unser Netzwerk, advozierte Größen mit sehr detaillierten, realistischen und zeitlich kohärenten Merkmalen zu erzeugen. Unsere Methode funktioniert sofort und verwendet nur einen einzigen Zeitschritt von niedrig aufgelösten Flüssigkeitsdaten. Wir demonstrieren die Fähigkeiten unserer Methode anhand einer Vielzahl komplexer Eingaben und Anwendungen in zwei und drei Dimensionen.

Um die Effizienz des Modelltrainings mit 3D-Volumendaten zu verbessern, greifen wir die klassische Idee des unbeaufsichtigten Autoencoder-Vortrainings wieder auf und schlagen eine modifizierte Variante vor, die auf einem vollständigen Rückwärtsdurchlauf basiert, der in Verbindung mit einer bestimmten Trainingsaufgabe trainiert wird. Dies führt zu Netzwerken, die so invertierbar wie möglich sind und gegenseitige Informationen über alle eingeschränkten Schichten hinweg teilen. Darüber hinaus stellen wir eine Verbindung zwischen der Singularwertzerlegung und dem Vortraining her und zeigen, wie diese genutzt werden kann, um Einblicke in die gelernten Strukturen zu gewinnen. Am wichtigsten ist, dass wir zeigen, dass unser Ansatz eine verbesserte Leistung für eine

Vielzahl von relevanten Lern- und Transferaufgaben liefert, die von vollständig verbundenen Netzwerken über residuelle neuronale Netzwerke bis hin zu GANs reichen.

Für natürliche Videoanwendungen konzentrieren wir uns auf die posengeleitete Generierung menschlicher Videos. Anstatt die menschliche Sequenz direkt zu generieren, schlagen wir einen neuen Ansatz vor, um zeitlich kohärente UV-Koordinaten zu erzeugen. Unsere Methode ist nicht an die Umrisse des menschlichen Körpers gebunden und kann lose Kleidungsstücke und Haare erfassen. Wir haben eine differenzierbare Pipeline implementiert, um die UV-Zuordnung zwischen einer Sequenz von RGB-Eingaben und Texturen über UV-Koordinaten zu lernen. Anstatt die UV-Koordinaten jedes Einzelbildes separat zu behandeln, verbindet unser Ansatz zur Datengenerierung alle UV-Koordinaten durch Feature-Matching, um zeitliche Stabilität zu gewährleisten. Anschließend wird ein generatives Modell trainiert, um die räumliche Qualität und zeitliche Stabilität auszugleichen. Es wird durch überwachte und nicht überwachte Verluste sowohl im UV- als auch im Bildraum gesteuert. Unsere Experimente zeigen, dass die trainierten Modelle hochwertige UV-Koordinaten ausgeben und sich auf neue Posen verallgemeinern lassen. Sobald eine Sequenz von UV-Koordinaten von unserem Modell abgeleitet wurde, kann sie verwendet werden, um flexibel neue Looks und veränderte visuelle Stile zu synthetisieren. Im Vergleich zu bestehenden Methoden reduziert unser Ansatz den Rechenaufwand für die Animation neuer Outfits um mehrere Größenordnungen.

Unsere Methoden erreichen eine räumlich realistische und zeitlich kohärente Sequenzgenerierung mit GANs. Unsere verschiedenen Ergebnisse, z. B. die Erzeugung von Flüssigkeitsströmen und natürlichen Sequenzen, zeigen die Leistungsfähigkeit von GANs bei der Videogenerierung. Wir hoffen, dass unsere Arbeit als Inspiration für andere verwandte Arbeiten dienen kann.

Contents

List of Figures

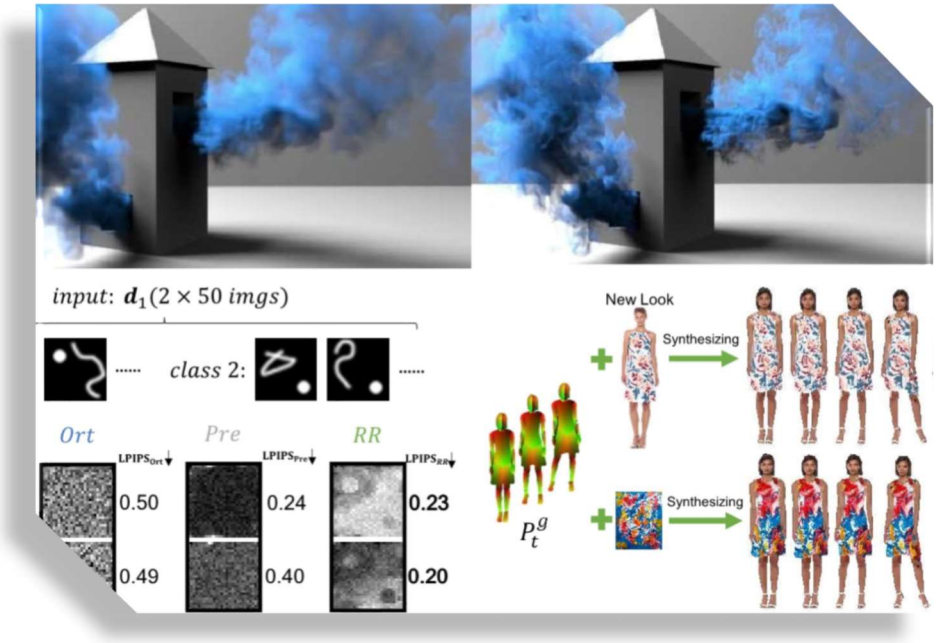
List of Tables

Acronyms

APIC	Affine PIC.
CNNs	Convolutional Neural Networks.
FLIP	Fluid-Implicit-Particle.
GANs	Generative Adversarial Networks.
GPU	Graphics Processing Units.
HR	high resolution.
LR	low resolution.
MPM	Material Point Method.
PCG	preconditioned conjugate gradient.
PIC	Particle-In-Cell.
ReLU	Rectified Linear Unit.
SPH	Smoothed-particle hydrodynamics.
SR	super-resolution.
VAE	Variational Auto-Encoder.

Part I

Introduction



"In motion alone is the answer
to all of the mysteries of matter."

Walter Russell

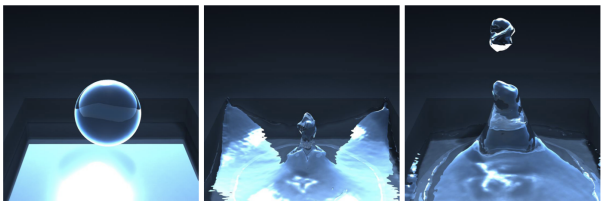
1 Introduction

The development of imaging technologies brings an explosive increase of digital data, such as images and videos. Compared with a static image, a video is a sequence of temporally coherent images and can record the movement of objects, which contains more dynamic information and is more aligned with what humans see in the real world. With those advantages, videos play more and more important roles in various situations, from daily life to mass entertainment to researches. Besides, in the field of computer graphics, which aims for digitally synthesizing and manipulating visual content, it's also crucial to develop video synthesizing algorithms. Traditionally, videos are typically synthesized via solving dynamic equations or motion tracking to generate realistic motions, as shown in Figure 1.1. Despite the rapid developments of computing hardware and algorithms, it's

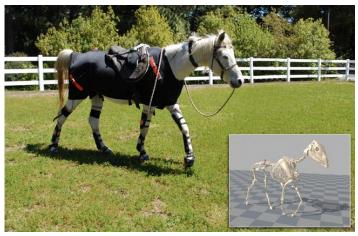
still highly challenging to efficiently generate videos with realistic dynamics, because of the complicated physical relationships between frames. On the other hand, deep learning methods achieve rapid developments and great success in the last decades in many fields, such as computer vision and natural language processing. Under this background, methods based on Convolutional Neural Networks (CNN) [1] provide a different approach for video synthesizing. Instead of generating the sequence via solving dynamic equations or tracing from real motion, CNN-based methods train a model to learn the dynamics from the dataset directly. Once the model is well trained from the dataset, then the model can be applied to generate sequences with learned corresponding dynamics.

Video generation shows great application prospects in various fields. Hollywood consumes a lot of human and material resources every year in video generation for good movies, e.g., a single "Game of Thrones" episode costs around 10 million dollars with a huge group of experts for visual effects. Video games, as shown in Figure 1.2 a, the market of which is above 100 billion dollars, are also highly relied on video generation technologies to improve users' experience. Another example is the Metaverse (Figure 1.2 b), which aims at building a virtual social world. Global market of Metaverse accounted for over 330 million dollars in 2020 and is estimated to be over 1000 million dollars by 2030. In the virtual Metaverse, scenes should be synthesized as realistic as our real world, so that users can be fully immersed in the virtual world and get the best user experience. To achieve this goal, video synthesizing technologies are indispensable.

Video generation technologies have been developed for decades. Except for solving dynamic equations or motion tracking to generate realistic motions, early works achieved video generation via synthesizing dynamic texture [2, 3, 4, 5]. While it is difficult to be applied in complicated situations. The machine learning based methods show the possibility to improve the efficiency of video generation. CNN is built with convolutional layers, then model size and non-linearity can be easily changed via modifying model layers. Besides, convolutional operations can be efficiently executed with Graphics Processing Units (GPU), which largely increases the computation efficiency. Those reasons result in CNN achieving great success in various fields and those are also what video generation needs. [6, 7, 8, 9] applied Variational Auto-Encoder (VAE) for video generation. However, those methods primarily applied traditional L-Norms loss functions, which aim at decreasing average loss values over the whole space and result in smoothed detail. Instead, GANs [10] aim at matching the output distribution with ground truth, which largely alleviates the problem of detail smoothing. With the development of GANs, the quality of the generated videos are largely improved, e.g., MocoGAN [11] achieved video generation via decomposing video content and dynamic motions, but resolution of generated video is quite limited since MocoGAN used a whole sequence as the inputs of the discriminator. On the other hand, Vid2vid [12] used data pair, image and optical flow, as the inputs of the discriminator, and achieved mapping between semantic videos and target videos, then new videos can be generated with corresponding semantic sequences. However, dynamic processing is not involved in the adversarial training, and discriminator cannot discover the correct dynamic relationship between adjacent frames. Thus, improving the temporal coherence of the generated videos is still challenging. This



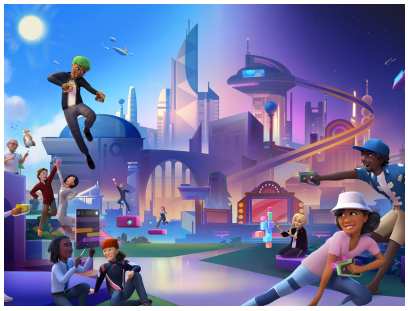
(a) Generating fluid sequence via solving dynamic equations [13]



(b) Motion capturing with sensors [14]

Figure 1.1: Examples of fluid simulation and motion tracking.

(a) Example of games using Unreal engine [15].



(b) Example of metaverse.

Figure 1.2: Example applications of video generation.

dissertation focuses on improving the temporal coherence of the videos generated from GANs.

1.1 Dissertation Overview

In this dissertation, we start by introducing the motivation of video generation in Part I. Then we explore approaches with applying GANs to improve efficiency for fluids simulation (part II) and natural videos generation(part III). Further discussion is provided in Part IV to summarize the connections between generating fluid sequence and natural sequence. An overview of the dissertation structure is shown in Figure 1.3. At the beginning of this chapter, a teaser image is presented as a preview of the results generated with methods proposed in this dissertation.

Fluid simulation aims at recovering realistic fluid motion, then those data can be applied for various purposes, e.g., explosion and fire scenes can be generated via smoke simulation for movies, data of which are extremely expensive, or dangerous, or even impossible, to be personally captured from the real scene. However, traditional technologies for fluid simulation require not only tremendous time cost, but considerable human and material resources to achieve the required quality. On the other hand, we found that it would be much more efficient to generate desired fluid data at low resolution (LR) than high resolution (HR). Thus, in Chap. 3, we present a generative model for fluid super-resolution (SR), generating HR data based on LR data. Preparing a collection

of paired low- and high-resolution data, our method bridges the gap between low- and high-resolution data via using HR data as ground truth to train a GAN model with LR data as generator input. However, training the network with traditional adversarial functions can only recover the sharp spatial details in the results, and for sequential data, temporal coherence is also an important aspect for quality evaluation. Thus, except for the traditional spatial discriminator, we proposed a novel temporal discriminator to improve the temporal coherence of the results, and we demonstrated that trained models can efficiently generate promising HR data with inputting LR data. However, training with fluid data is time-consuming because of large data size and model should be retrained from scratch for new fluid scenes. On the other hand, all the fluid simulation data can be generated with the same dynamic equations, e.g., Navier-Stokes equations, which means there are implicit connections for different scenes. In Chap. 4, we propose a novel training pipeline to make the CNN learn dominant features from the data set, so that learned model can be applied for different fluid scenes as transfer learning. Unlike traditional CNN, which only builds the forward pipeline from input to output, we also build the backward pipeline with shared weights from the forward pipeline, and using reconstruction loss during training to make CNN extract dominant features from the data set.

Natural video generation brings great convenience to our daily life. For instance, buying shoes or clothing frequently happens in our daily life, but normally we spend lots of time for try-on to see whether the style fits or not. Traditional methods normally can deal with clothes that fit close to the human body, but loose clothing is still challenging. For instance, DensePose [16] can generate UV coordinates from a single image only for the human body part. Besides, DensePose aims for single image representation, and temporal incoherence happened when applying DensePose for videos. In Chap. 5, we present our approach temporalUV, which can generate temporally coherent UV coordinates for videos with loose clothing. The most challenging step of generating temporally coherent UV coordinates for loose clothing is the lack of ground truth data, which means that the supervised training approach would not be valid in this case. On the other hand, high non-linearity in the transformation between images and textures makes it also difficult to successfully train a model with only un-supervised training. Thus, our approach combines both supervised and un-supervised losses for training the model. Firstly, we build a differentiable pipeline to achieve transformation between images and textures with UV coordinates, which allows us to fill the missing UV coordinates for the raw UVs, e.g., UVs generated from the DensePose model or SMPL model, via extrapolation and optimization. Then we applied temporal relocation to improve the temporal coherence of the UVs sequence. Those pre-processed data are regarded as guidance for supervised training. We also apply the differentiable transformation pipeline during training, so that images can be generated from UVs and texture, and un-supervised losses from the images can be applied to improve the resulting UVs quality. We use different matrices and conduct user studies to demonstrate that our approach achieves significant improvements than baselines.

Except works discussed above, we also explore the GAN approaches to achieve fluid super-resolution with larger-scale factors, such as 8, and we also applied spatial-temporal

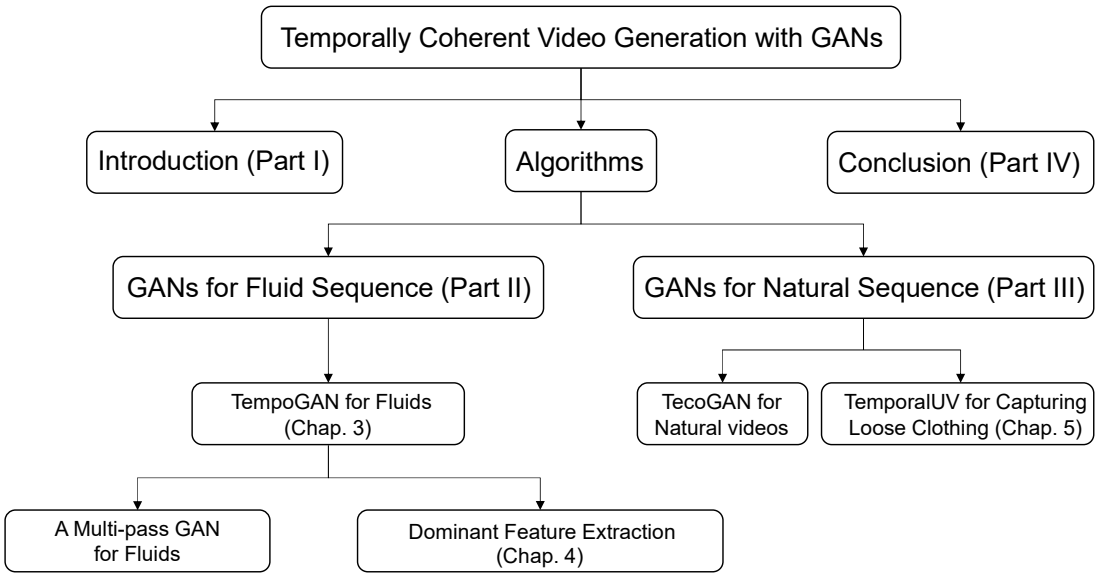


Figure 1.3: An overview of the dissertation structure.

adversarial learning for more natural video applications, such as video SR and unpaired video translation. Discussion about those parts is given in Chap. 7 to gain a more in-depth understanding of video generation with GANs. Furthermore, we summarize the contributions, current challenges, and potential future directions in the last part of this dissertation.

1.2 Publication List

This dissertation explains and concludes researches from the following manuscripts:

Publications:

1. **Y. Xie***, E. Franz*, M. Chu*, and N. Thuerey, "tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 95, 2018

2. M. Werhahn, **Y. Xie**, M. Chu, and N. Thuerey, "A multi-pass GAN for fluid flow super-resolution," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 2, no. 2, Jul. 2019

3. M. Chu* , **Y. Xie***, J. Mayer, L. Leal-Taixé and N. Thuerey, "Learning temporal coherence via self-supervision for GAN-based video generation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, p. 75, 2020

4. **Y. Xie**, H. Mao, A. Yao and N. Thuerey, "TemporalUV: Capturing Loose Clothing with Temporally Coherent UV Coordinates", In Proceedings of the IEEE/CVF Conference on *Computer Vision and Pattern Recognition*, pp. 3450-3459. 2022

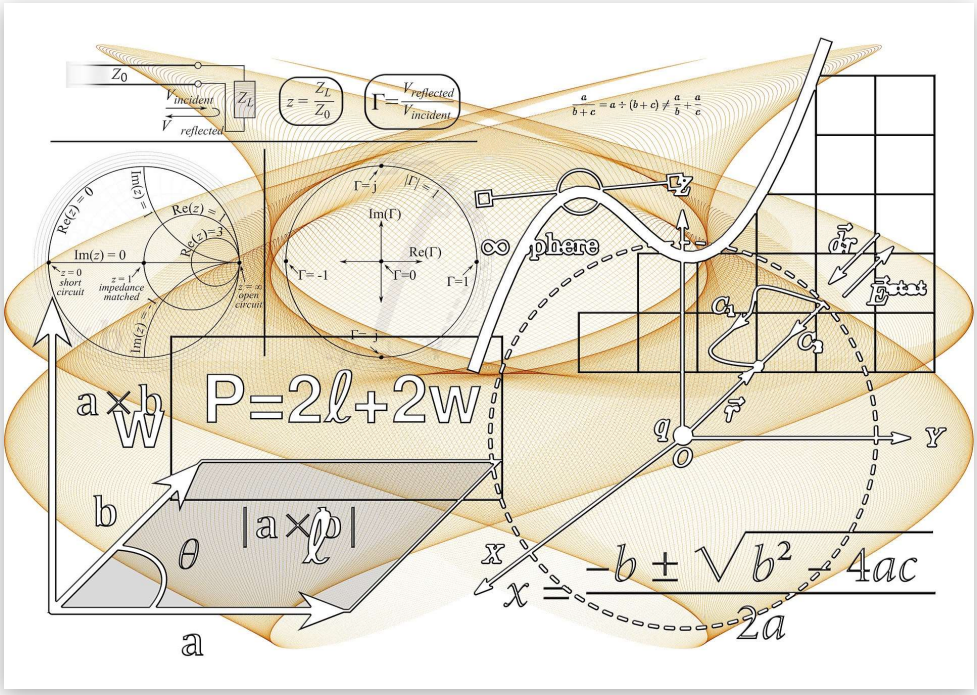
5. **Y. Xie** and N. Thuerey, "Reviving Autoencoder Pretraining", Neural Comput Applic 35, 4587–4619 (2023). https://doi.org/10.1007/s00521-022-07892-0

6. Z. Yu, L. Yang,**Y. Xie**, P. Chen, A. Yao. "UV-Based 3D Hand-Object Reconstruction with Grasp Optimization", *the 33rd British Machine Vision Conference* (Spotlight), 2022.

Pre-prints:

1. N. Thuerey, **Y. Xie**, M. Chu, S. Wiewel, and L. Prantl, "Physics-Based Deep Learning for Fluid Flow," NeurIPS Workshop, Modeling the Physical World, 2018.

*: These authors contributed equally to the paper



"You must have a solid foundation
if you're going to have a strong superstructure."

Gordon B. Hinckley

2 Fundamentals

A journey of a thousand miles begins with a single step. Video generation, as a developing area, is still facing various challenges which hinder performance improvement. For instance, fluid simulation suffers from numerical dissipation in the advection step and computational cost during the projection step; the balance between spatial quality and temporal coherence is still non-trivial for natural sequence generation. In this chapter, we will first introduce fundamental knowledge about fluid simulation and corresponding difficulties during simulation steps. Then machine learning algorithms, especially GANs,

will be outlined. In the end, we will discuss related works, which applied machine learning algorithms to solve the problems during video generation.

2.1 Fluid Simulation

Fluid simulation is typically generated via solving Navier-Stokes equations, which can be written as:

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} &= -\frac{1}{\rho}\nabla p + v\nabla \cdot \nabla \boldsymbol{u} + \boldsymbol{g}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned} \tag{2.1}$$

Here, $\boldsymbol{u}$, p and v represent velocity, pressure, and kinematic viscosity, respectively. $\boldsymbol{g}$ means the acceleration due to external forces, such as gravity. Incompressible Navier-Stokes equations contain two parts: the momentum equation (first row), which is derived from Newton's second law and indicates how the external forces influence the fluid; and the incompressibility condition (second row), which preserves the volume of the flow. The momentum equation can also be written with the material derivative:

$$\begin{aligned} \frac{D\boldsymbol{u}}{Dt} &= -\frac{1}{\rho}\nabla p + v\nabla \cdot \nabla \boldsymbol{u} + \boldsymbol{g}, \\ \text{with} \quad \frac{D\boldsymbol{u}}{Dt} &= \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}. \end{aligned} \tag{2.2}$$

In this dissertation, we primarily focus on the single-phase inviscid flow. Thus, the viscosity term is not considered and Equation 2.2 will be simplified as *Euler* equations:

$$\begin{aligned} \frac{D\boldsymbol{u}}{Dt} &= -\frac{1}{\rho}\nabla p + \boldsymbol{g}, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned} \tag{2.3}$$

How to solve Equation 2.2 mathematically is still an open challenge, because those partial differential equations are highly non-linear, and in particular, turbulence is normally included in the solutions, which is also an unsolved problem. Thus, Equation 2.2 is typically solved numerically to get the fluid simulation results [17]. To generate visually realistic fluid simulations, [18] firstly brought a stable single-phase fluid simulation algorithm to computer animation. Based on this algorithm, extension works and developed algorithms were proposed. E.g., [19, 20] proposed more accurate advection schemes; [21, 22] researched the coupling between fluid and solid, so that interaction simulation between fluid and obstacles becomes possible. Other popular works, such as [23, 24], increase flows' resolution by adding turbulence to the flow. Numerical fluid solver can be classified into different categories according to the viewpoint to track the fluid motion: *Eulerian* methods, *Lagrangian* methods and *Hybrid* methods.

2.1.1 Eulerian Methods

Eulerian methods treat the flow as a continuous phase and are mesh-based methods, such as uniform grids. For Eulerian methods, we can assume that observers are standstill on

the grids and record all fluid parameters step by step. At every time step, fluid parameters will be updated with a finite volume scheme on the mesh grids. Normally Eulerian method based fluid solver solves Equation 2.3 with three split parts: the advection part

$$\frac{D\boldsymbol{u}}{Dt} = 0, \tag{2.4}$$

the body forces part

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{g}, \tag{2.5}$$

and the pressure projection part

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\frac{1}{\rho}\nabla p, \quad \text{s.t. } \nabla \cdot \boldsymbol{u} = 0. \tag{2.6}$$

The advection function computes how the transportation influences the quantities with the velocity field $\boldsymbol{u}$ for a time interval. Solving Equation 2.4 with direct Forward Euler discretization is unconditionally unstable and the flow will eventually blow up. The semi-Lagrangian method is commonly used to avoid this problem, which applies a Lagrangian viewpoint with imaginary particles. Given position x_A and its velocity $\boldsymbol{u}^n$ at time step n, semi-Lagrangian aims at looking for a particle located at x_B, which will end up at x_A after one time step Δt. This step is achieved with:

$$x_B = x_A - \boldsymbol{u}^n \Delta t. \tag{2.7}$$

Thus, fluid parameters at x_A will be replaced with x_B at time step $n+1$. x_B is most likely not located at the grid, then fluid parameters at x_B will be interpolated from the neighbor grids. This scheme is popular since it's unconditional stable, so it's allowed to use a large time step and is especially efficient for simulating large-scale flows.

However, interpolation applied in the semi-Lagrangian method brings another problem, numerical dissipation. Spatial interpolation results in the smoothing of the fluid parameters, which decreases the system energy. Thus, even viscosity is not considered in Equation 2.3, but generated fluid looks unnatural sticky. Furthermore, turbulence details in the flow will also be smoothed out by the numerical dissipation. Till now, it's hard to measure the numerical dissipation with mathematical expressions, while the complex flow is widely used in many areas, such as games and movies. Thus, a lot of works focus on improving numerical dissipation. Based on the semi-Lagrangian method, BFECC [19] and MacCormack [20] improve the accuracy via calculating the difference between forward and backward warping, i.e.,

$$\begin{aligned} x_B^* = x_A - \boldsymbol{u}^n \Delta t, \;\; x_B^{**} = x_B^* + \boldsymbol{u}^n \Delta t, \\ x_B^{error} = \frac{1}{2}(x_B^{**} - x_A), x_B = x_B^* + x_B^{error}. \end{aligned} \tag{2.8}$$

However, those methods cannot fundamentally avoid the accumulated errors. Instead, [25] achieved the fluid energy preservation discretely, but non-linearly coupled equations required to be solved. [26] proposed an energy-preserving reflection operator for

the projection step, which largely alleviated the energy reduction problem and led to improved detail preservation. But this method was only focused on the energy conservation of velocity dynamics. More recent work $BiMocq^2$ [27], as an unconditionally stable method evolved mapping functions to decrease numerical dissipation. However, $BiMocq^2$ required further improvements for the free-surface boundary conditions.

On the other hand, vorticity modeling is another category of the solution, which focuses on the curl form of the Navier-Stokes equation [28]. Different algorithms were proposed using vortex filaments [29], vortex sheets [30], vorticity amplification [31, 32], or anisotropic vortices [33].

After the advection step, the body force part Equation 2.5 calculates the velocity change caused by external forces, such as gravity. The third step is the pressure projection part, i.e., Equation 2.6. The pressure projection step aims at a divergence-free velocity field $\boldsymbol{u}$ under required boundary conditions. Divergence-free is an important feature of the incompressible flow, which guarantees the conservation of mass. To solve the pressure projection step, we can transform Equation 2.6 into:

$$\nabla \cdot \nabla p = \frac{\rho}{\Delta t} \nabla \cdot \boldsymbol{u}, \tag{2.9}$$

where $\boldsymbol{u}$ is assumed to be the velocity result from the body force step Equation 2.5. We can see that Equation 2.6 can be written as a Poisson equation with Equation 2.9. Then solving the pressure projection step is equivalent to solving a linear equation $Ap = b$, where A represents a large, sparse, and symmetric matrix. This linear system can be solved with preconditioned conjugate gradient (PCG) solver [34]. With solved pressure p, final velocity can be recomputed from Equation 2.9. However, solving Equation 2.9 with a PCG solver is computationally expensive, especially with large resolution, which leads the pressure projection step to the most time-consuming step among the whole solving procedure. Thus, various algorithms for accelerating the pressure projection step were proposed in the last decades, and they can be classified into two categories according to whether they aim for reducing algorithm complexity, such as [35, 36, 37, 38, 39], or the type of data structures, such as [40, 41, 42].

2.1.2 Lagrangian Methods

Lagrangian methods, on the other hand, treat the flow as a discrete phase or particles, and the state of the particles is updated via solving flow equations. For Lagrangian methods, we can assume that observers follow the movements of the particles and record all fluid parameters step by step. Smoothed-particle hydrodynamics (SPH) is the most commonly used Lagrangian fluid. Without interpolation during the advection step, SPH is good at linear and angular momentum conservation, but SPH computes fluid parameters for every particle purely with its neighbor particles, which cannot guarantee incompressibility and stability. Thus, different various of SPH were proposed to overcome those weaknesses, such as the weakly compressible SPH [43] and the implicit incompressible SPH [44].

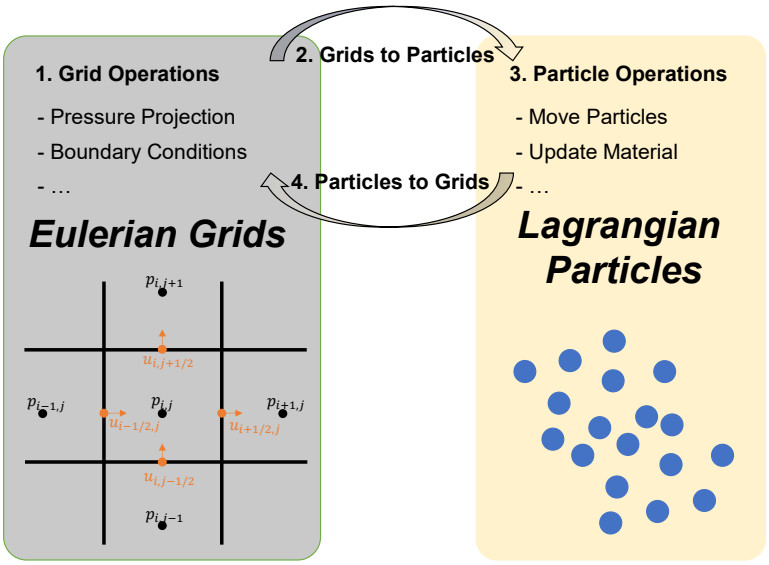


Figure 2.1: Hybrid method overview.

2.1.3 Hybrid Methods

With the discussions above, we can see that Eulerian methods are good at discretization, and it's efficient to achieve neighbor look-up with Eulerian grids. But it's hard to overcome the numerical dissipation problem with the Eulerian methods. On the contrary, Lagrangian methods are good at the advection step, which means lower numerical dissipation error and better energy conservation. However, without grids, it's difficult to achieve discretization with the Lagrangian method and complicated data structures are required for neighbor look-up. Thus, hybrid methods are proposed to combine those two viewpoints, using the Lagrangian viewpoint during kinematic steps, e.g., advection, but Eulerian grids for dynamic steps, e.g., pressure projection step. Overview of hybrid method is shown in Figure 2.1. It is worth pointing out that in the procedures of "Particles to Grid" and "Grid to Particles", kernel functions are applied to achieve appropriate transformation. E.g., during transferring the parameters from particles to the grids, normally if grids are closer to the particle, the particle should influence those grids more than others far away from the particle. Thus, kernel functions such as B-spline kernels can be applied so that particles will not treat all neighbor grids equally. Various hybrid methods were proposed with different transformation schemes between grids and particles. Early works, such as Particle-In-Cell (PIC) [45], transfer all fluid parameters from the grid to particles, which is stable but leads to obvious numerical dissipation. Fluid-Implicit-Particle (FLIP) method [46] improved the dissipation problem via purely applying interpolation for the change of the flow from the grid, but FLIP is less stable. Affine PIC (APIC) [47] and PoliPic [48] constructed extra functions to improve the numeric noise problem in FLIP and preserve the local features. Material point method (MPM) [49, 50, 51] also takes into consideration the deformation gradient.

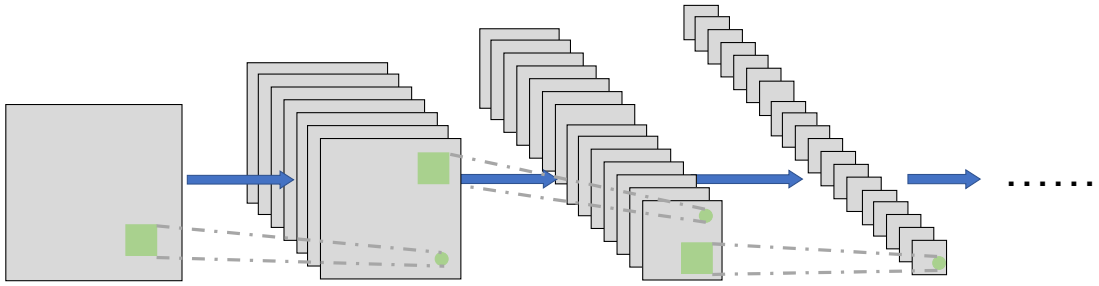


Figure 2.2: Example of 2D convolutional neural networks.

Since every particle in MPM can have a different material model, interactions between different phases can be conveniently addressed. MPM is commonly used for continuum material simulations, e.g., solids, fluids, or multi-phase interactions. Hybrid methods make use of the strengths of Eulerian and Lagrangian representations, but non-trivial transferring between grid and particles leads to extra computational cost.

2.2 Deep Learning for Sequence Generation

Deep learning techniques have achieved significant breakthroughs in numerous fields such as classification [52], object detection [53], style transfer [54], novel view synthesis [55], and additionally, in the area of content creation [56]. Sequence generation, which is the main focus of this dissertation, is also a subset of content creation. Deep learning generative model f can be regarded as a mapping from input domain X to the target domain Y, which can be summarized as:

$$Y = f(X|\theta), \tag{2.10}$$

where θ are trainable parameters of the model f. Input domain X could be 1-D vectors or 2-D images. The output of the model normally can be an image or a sequence directly. Training can be classified into supervised training if paired input and output data are given, which can be manually or automatically labeled. Otherwise, it's unsupervised training if the corresponding ground truth of the model output is not given. Then the model can be trained via minimizing designed loss functions. Minimizing the loss functions is normally equivalent to maximizing the likelihood function of the target dataset, or minimizing the Kullback–Leibler divergence between the model output and target dataset distribution. In this section, we first introduce the fundamentals of deep learning methods, such as neural networks. Then we will continue with introducing the adversarial training approach, GANs. After that, a review on applying GANs for fluid simulation and natural sequence will be given.

2.2.1 Neural Networks and CNNs

Neural networks are composed of artificial neurons, and neurons at different layers are connected via trainable weights W, bias b, and activation function σ, which can be

summarized as:

$$y_{i+1} = \sigma(W y_i + b), \tag{2.11}$$

where y_i and y_{i+1} are from the i_{th} and $(i+1)_{th}$ layers, respectively. The first layer of the network is the input data layer and output is regarded as the last layer. Operation $Wy_i + b$ achieves linear transformation of y_i, and activation function σ brings non-linear transformation to y_i. Practically, mapping from input to output distribution is highly non-linear, so σ plays a very important role to make the model well trained, e.g., Rectified Linear Unit (ReLU) function ($\sigma_{ReLU}(z) = max(0, z)$) keeps all values in the output to be non-negative; sigmoid function ($\sigma_{sigmoid}(z) = \frac{1}{1+e^{-z}}$) and Tanh function ($\sigma_{Tanh} = Tanh(z)$) restrict all the values in the range [0,1] and [-1,1], respectively. Thus, network capacity can be modified via adjusting the number of layers, weights, and activation functions. And theoretically, all logical functions can be modeled with neural networks. Besides, all operations in Equation 2.11 are differentiable, which brings the possibility of training the network with gradient-based optimizers.

Once the model is built and data are prepared, trainable parameters, such as weights and bias, in the neural network can be trained via minimizing designed loss functions. Proper loss functions are crucial for training. Loss functions normally contain terms, which can measure the distance between neural network outputs and corresponding ground truth, and also regularization terms, which aim at avoiding model over-fitting. Training proceeds via gradient back-propagation. For every step, the gradient of the loss functions with respect to the trainable parameters can be computed via chain rules. Then weights can be updated according to the gradient. The traditional gradient descent method moves the weights along the direction of the gradient with the learning rate as the moving distance. However, frequent updating of the trainable parameters results in the oscillation of the loss function and it's easy to fall into a local minimum point or saddle point when training with the gradient descent method. RMSprop, Adagrad, Adam algorithms bring momentum for every step optimization, which can stabilize the weight updating direction and adaptively change the learning rate. However, even training with momentum and adaptive learning rate, global optimal solutions are not guaranteed, because the loss functions are normally highly non-linear and non-convex.

Images are typically analyzed with various filters, e.g., information at different frequencies can be separated with a corresponding frequency filter; derivatives of the images can be computed with derivative kernels. Those filters can be conveniently represented with convolutional operations. For Equation 2.11 of the neural networks, we can also replace the matrix multiplication Wy_i with convolutional operations to achieve CNNs. CNNs have been applied in numerous eulerian representation related tasks, such as images, videos. CNNs only need a small number of parameters in every layer as a local-wise kernel for feature extraction, and the same kernel can be applied in the whole input image, which largely decreases the number of training parameters and improves the training efficiency. With decreased number of parameters, CNNs also benefit from the reduced risk of over-fitting. Convolutional layers are normally combined with pooling layers, which can decrease the dimension for the next layers, and also make it possible

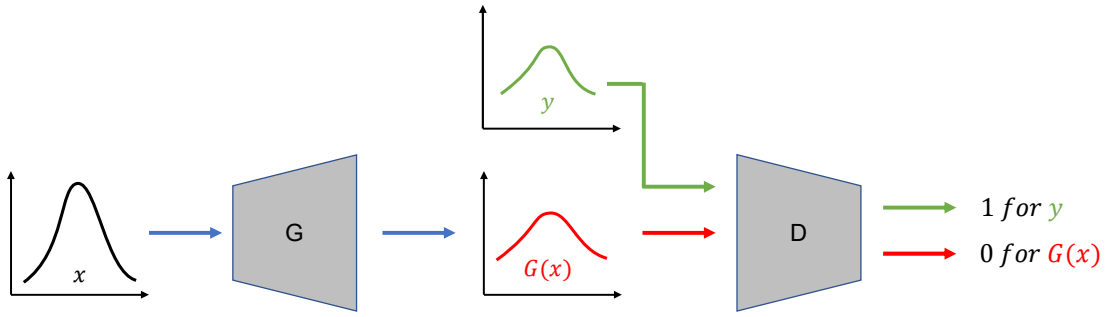


Figure 2.3: Pipeline overview of GANs. Generator G aims at reproducing target distribution y from input distribution x. At every step, discriminator D is trained to distinguish real distribution y and estimated distribution $G(x)$ from the generator, while G is trained to fool D via improving the quality of $G(x)$. After the model is trained, the generator can output results that are close to the target distribution y and the discriminator cannot distinguish $G(x)$ and y anymore.

to build connections for points far away from each other even with small size kernels. An example 2D CNN is shown in Figure 2.2.

2.2.2 GANs

In this section, we will introduce one of the popular methods to generate content, GANs [57], which is also the main focus of this dissertation. They were shown to be particularly powerful at re-creating the distributions of complex data sets, such as images of human faces. GANs consist of two models, which are trained in conjunction: the generator G and the discriminator D. Both are typically realized as CNNs. For regular ones, i.e., unconditional GANs, the goal is to train a generator $G(x)$ that maps a simple data distribution, typically noise, x to a complex desired output y, e.g., natural images. discriminator D is trained to classify whether its input is from the true data distribution or not.

Instead of using a manually specified loss term to train the generator, another NN, the discriminator, is used as a complex, learned loss function [57]. This discriminator takes the form of a simple binary classifier, which is trained in a supervised manner to reject *generated* data, i.e., it should return $D(G(x)) = 0$, and accept the *real* data with $D(y) = 1$. For training, the loss for the discriminator is thus given by a sigmoid cross-entropy for the two classes "generated" and "real". On the contrary, generator is trained to achieve $D(G(x)) = 1$. Loss functions of discriminator and generator can be summarized as:

$$\begin{aligned} \mathcal{L}_D &= \mathbb{E}_{y \sim p_y(y)}[-\log D(y)] + \mathbb{E}_{x \sim p_x(x)}[-\log(1 - D(G(x)))], \\ \mathcal{L}_G &= \mathbb{E}_{x \sim p_x(x)}[\log(1 - D(G(x)))]. \end{aligned} \tag{2.12}$$

Overview of the GANs structure is shown in Figure 2.3. Generator and discriminator are trained together. Theoretically, the training will arrive at a convergent stage when

generator and discriminator achieve an equilibrium state and under this situation, the discriminator cannot distinguish the generated data $G(x)$ from the true data y, which means that GANs are trained via matching the distribution of $G(x)$ with y. With this training approach, GANs show state-of-the-art performance with multi-modal problems. The problem of multi-modality means that for every input x, the valid target y is not unique, such as super-resolution. This is a pretty common problem in generative tasks. For non-adversarial training approaches, normally L-norm-based losses are applied to measure the distance between $G(x)$ and y, then the model will be trained via minimizing the distance for the whole dataset. However, data samples in different modalities may cause conflicted gradients feedback for the optimization and result in the differences between distributions $G(x)$ and y. From the point of view of the generated images, we will see that details are smoothed in the results. On the contrary, GANs training directly minimizes the distance between distributions $G(x)$ and y, rather than computing the difference for every data sample, which yields better perceptual quality.

The generator is not trained with manually designed loss functions, but $\mathcal{L}_G$ in Equation 2.12 is still equivalent to minimizing the Jensen-Shannon divergence between $G(x)$ and y. One challenge for applying Equation 2.12 to train GANs is how to balance $\mathcal{L}_D$ and $\mathcal{L}_G$, e.g., if the discriminator is trained better than the generator, then $\mathcal{L}_D$ would be small, which results in gradient vanishing problem for the generator and makes it hard to improve the generator further. Several variants are proposed to alleviate this problem, e.g., Wasserstein GAN [58]

$$\begin{aligned} \mathcal{L}_D &= -\mathbb{E}_{y\sim p_y(y)}[D(y)] + \mathbb{E}_{x\sim p_x(x)}[D(G(x))], \\ \mathcal{L}_G &= -\mathbb{E}_{x\sim p_x(x)}[D(G(x))]. \end{aligned} \tag{2.13}$$

However, the weights of the discriminator of the Wasserstein GAN are restricted within a certain range, which largely limits the capacity of the discriminator. Thus, gradient penalty is applied in Wasserstein GAN to improve the network property [59]:

$$\begin{aligned} \mathcal{L}_D &= -\mathbb{E}_{y\sim p_y(y)}[D(y)] + \mathbb{E}_{x\sim p_x(x)}[D(G(x))] + \lambda\mathbb{E}_{\hat{y}\sim p_{\hat{y}}(\hat{y})}[(\nabla D(\hat{y}) - 1)^2], \\ \mathcal{L}_G &= -\mathbb{E}_{x\sim p_x(x)}[D(G(x))], \\ \hat{y} &= \epsilon y + (1-\epsilon)G(x). \end{aligned} \tag{2.14}$$

Other variants to solve the gradient vanishing problems are like least-square GAN [60] and relativistic average GAN [61]. It's worth pointing out that vanilla GANs apply Sigmoid activation in the last layer of the discriminator, while Wasserstein GAN, least-square GAN, and relativistic average GAN do not apply any activation functions in the last layer of the discriminator.

Another common problem for GANs training is model collapse. When the generator generates a result $\hat{y}$ with a high discriminator score $D(\hat{y})$, theoretically, the discriminator should learn to reject $\hat{y}$ with a low score. However, if the next generation of the discriminator gets stuck in a local minimum, and continuously outputs a high score for $\hat{y}$, then it's easy for the generator to always output $\hat{y}$ since the task of the generator is to generate the most plausible output for the current discriminator. Based on the vanilla

GANs, amount of strategies are proposed to balance generator and discriminator during the training. Progressive growing GAN [62] achieved stabilized training via starting the generative task with a small resolution and then increasing the output resolution and model size progressively. [63] and [64] applied various regularizations to improve GANs convergence.

Depending on the kind of input data they take, GANs can be separated into unconditional and conditional ones. The formers generate realistic data from samples of a synthetic data distribution like Gaussian noise. The DC-GAN [65] is a good example of an unconditional GAN. It was designed for generic natural images, while the cycle-consistent GAN by Zhu et al. [66] was developed to translate between different classes of images. The conditional GANs were introduced by Mirza and Osindero [67], and provide the network with an input that is in some way related to the target function to control the generated output. Therefore, conditional variants are popular for transformation tasks, such as image translations problems [68] and super-resolution problems [69].

2.2.3 Deep Learning for Fluid Sequences

Deep learning algorithms have begun to influence computer graphics algorithms over the last decades. E.g., they were successfully used for efficient and noise-free renderings [70, 71], the illumination of volumes [72], modeling porous media [73], and character control [74]. First works also exist that target numerical simulations. E.g., a conditional GAN was used to compute solutions for smaller, two-dimensional advection-diffusion problems [75, 76]. Others have demonstrated the inference of SPH forces with regression forests [77], proposed CNNs for fast pressure projections [78], learned space-time deformations for interactive liquids [79]. [80] used CNN to find out the relationship between control parameters and simulations of interactive liquids. [81] applied a neural network model to help with the splashes generation. [82] used GANs to build the relationship between the solver's parameters and velocity field directly.

Overall, the methods discussed above are aimed at efficiently generating highly detailed fluid simulation, and another type of solution for this goal is fluid super-resolution. Since HR fluid simulation is much more time-consuming than LR simulation, it would be very efficient if we could use a super-resolution algorithm to generate HR data based on LR data. Chu et al. [83] proposed a method to lookup pre-computed patches using CNN-based descriptors. However, their method still required additional Lagrangian tracking information, and need to store a large amount of data at runtime apart from the trained model.

In this dissertation, we primarily explore GAN-based methods for fluid super-resolution. Regular GANs losses offer good performance for image generation tasks in terms of PSNR metrics [84, 85, 86]. However, sequence generation requires the generation of realistic content that changes naturally over time. Temporal coherence is a crucial aspect for video generation tasks, and it is especially so for conditional video generation tasks [87, 88, 89, 90], where specific correlations between the input and the generated spatio-temporal evolution are required. While discriminators of GANs typically only supervise the spatial content. Thus, directly applying GANs without carefully engineered con-

straints typically results in strong artifacts over time due to the significant difficulties introduced by the temporal changes. E.g., [91] focused on images without temporal constraints and generators can fail to learn the temporal cycle-consistency for videos.

To improve the spatial details and temporal coherence for video generation tasks, one way to solve this problem is by directly incorporating the time axis [92, 93, 94, 95], i.e., by using sequences of data as input and output. E.g., [96] proposed a temporal generator in their work; [97] recurrently used previously estimated outputs;.[98] proposed a sequence generator that learns a stochastic policy. However, for fluid simulation, large data size limits the algorithms to take multiple frames as generator's inputs or outputs. An alternative method is to generate single-frame data with additional loss terms to keep the results coherent over time. While L1 and L2 temporal losses based on warping are generally used to enforce temporal smoothness in video generation tasks [99, 100, 101, 102], it leads to an undesirable smoothing over spatial detail and temporal changes in outputs. [103] achieved improved coherence in their results for video frame prediction, by adding specially designed distance measures as a discontinuity penalty between nearby frames. Similarly, [104] used neural networks to learn spatial alignment for low-resolution inputs, and adaptive aggregation for high-resolution outputs, which also improved the temporal coherence.

2.2.4 Deep Learning for Natural Sequences

Sharing similarities with deep learning for fluid sequence, improving spatial details and temporal coherence are also key points to natural sequence generation tasks, which are discussed in detail in the last section. In this dissertation, we primarily focus on pose guided human sequences generation. In image or video generation tasks [105, 106] that involve people, it is crucial to obtain accurate representations of the 3D human shape and appearance that allow for the efficient generation of modified content. *UV coordinates* are a popular 2D representation in this context that establishes dense correspondences between 2D images and 3D surface-based representations of the human body. UV coordinates go beyond skeleton landmarks as a means for encoding human pose and shape, and are widely used to achieve realistic and high-quality generations [107, 108, 109]. Human body UV coordinates can be derived indirectly from estimates of 3D shape models [110, 111, 112, 113] like SMPL [114]. Alternatively, direct estimation methods like DensePose [16] and UltraPose [115] bypass intermediate 3D models to directly output UV coordinates from a single RGB image. The convenience of direct methods has led to DensePose being widely used in animation and editing applications [109, 116, 117, 118]. Irregardless, the UV coordinates obtained from SMPL and DensePose approximate only tight human body silhouettes. They do not capture loose clothing, such as long skirts or wide pants. In addition, the methods for UV estimation work only on individual images; for video inputs, they are applied frame-by-frame [119, 120], without considering the temporal relationship between frames. As such, the UV coordinates are inconsistent over time, so any re-targeted sequences will shift and jitter.

Pose-guided methods [121, 122, 123, 124, 109, 107, 108] generate images of a person with designated target poses. The appearance information is sourced from either images (image-to-video [125, 120, 126, 127]) or videos (video-to-video [128, 129, 130, 131]).

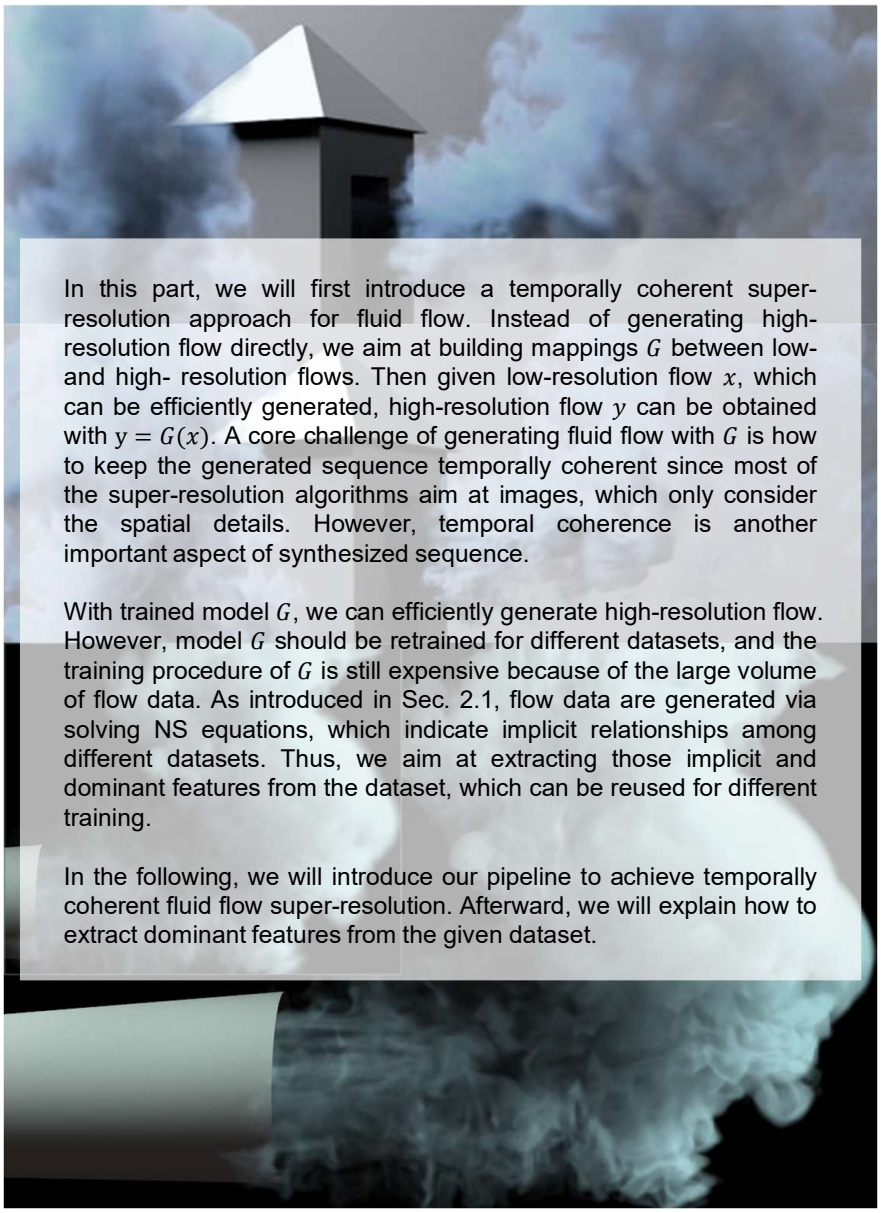
Image-to-Video: An early example is MonkeyNet [125]; while Monkeynet decouples appearance and motion information, it uses keypoints, which is insufficient for high quality capture of human body or clothing with complex textures. DwNet [120] used DensePose UV coordinates as inputs. DwNet applies an encoder-decoder architecture that warps the human body from source to target poses. However, DwNet can be difficult to train due to its use of highly non-linear warping grids. The generator also needs to be re-run for computing the warping grid each time the source image changes.

Video-to-Video: These methods [128, 129, 130, 131] have access to a source video and can therefore create richer models of the source subject than single image sources. In particular, [130] generate videos with spatial transformation of target poses, allowing it to capture loose clothing. However, all these works rely on 2D keypoints, making it hard to consider complicated visual styles.

Generalized Video Generation: Early methods modeled the entire video clip as a single latent representation [132, 96]. Follow-up work MoCoGAN [11] used a disentangled representation, separating appearance and motion. However, the model is not conditional, so cannot generate videos conditioned e.g. on target appearances or motions. End-to-end video re-targeting works RecycleGAN[133] and Vid2Vid[12] generate videos with content from one source video and motion from another, while a concurrent work [134] chose to learn motion translation in addition to the spatial content translation. These methods train target-specific models, in that a new network is trained for each target video.

Part II

Fluid Sequence Generation with GANs



In this part, we will first introduce a temporally coherent super-resolution approach for fluid flow. Instead of generating high-resolution flow directly, we aim at building mappings G between low- and high- resolution flows. Then given low-resolution flow x, which can be efficiently generated, high-resolution flow y can be obtained with $y = G(x)$. A core challenge of generating fluid flow with G is how to keep the generated sequence temporally coherent since most of the super-resolution algorithms aim at images, which only consider the spatial details. However, temporal coherence is another important aspect of synthesized sequence.

With trained model G, we can efficiently generate high-resolution flow. However, model G should be retrained for different datasets, and the training procedure of G is still expensive because of the large volume of flow data. As introduced in Sec. 2.1, flow data are generated via solving NS equations, which indicate implicit relationships among different datasets. Thus, we aim at extracting those implicit and dominant features from the dataset, which can be reused for different training.

In the following, we will introduce our pipeline to achieve temporally coherent fluid flow super-resolution. Afterward, we will explain how to extract dominant features from the given dataset.

3 Temporally Coherent Fluid Super-Resolution

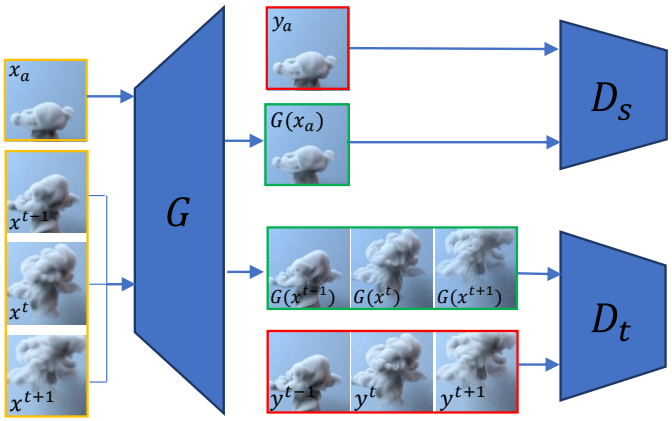


Figure 3.1: This figure gives a high-level overview of our approach: a generator on the left is guided during training by two discriminator networks (right), one of which focuses on space (D_s), while the other one focuses on temporal aspects (D_t). At runtime, both are discarded, and only the generator network is evaluated.

Capturing the intricate details of turbulent flows has been a long-standing challenge for numerical simulations. Resolving such details with discretized models induces enormous computational costs and quickly becomes infeasible for flows on human space and time scales. While algorithms to increase the apparent resolution of simulations can alleviate this problem [23], they are typically based on procedural models that are only loosely inspired by the underlying physics. In contrast to all previous methods, our algorithm represents a physically-based interpolation, that does not require any form of additional temporal data or quantities tracked over time. The SR process is instantaneous, based on volumetric data from a single frame of a fluid simulation. We found that inference of HR data in a fluid flow setting benefits from the availability of information about the flow. In our case, this takes the shape of additional physical variables such as velocity and vorticity as inputs, which in turn yield means for artistic control. A particular

challenge in the field of SR flow is how to evaluate the quality of the generated output. As we are typically targeting turbulent motions, a single coarse approximation can be associated with a large variety of significantly different HR versions. As long as the output matches the correlated spatial and temporal distributions of the reference data, it represents a correct solution.

A core challenge of generating flow data with deep learning methods is the temporal coherence of the results. Various SR algorithms are proposed for images [135, 136], which only consider the spatial quality. However, fluid flow, as dynamic sequences, should be temporally coherent and all the parameters should be changed smoothly. We propose a temporally coherent generative model addressing the SR problem for fluid flows. Our work represents the first approach to synthesizing four-dimensional physics fields with neural networks. Based on a conditional generative adversarial network that is designed for the inference of three-dimensional volumetric data, our model generates consistent and detailed results by using a novel temporal discriminator, in addition to the commonly used spatial one. Our experiments show that the generator is able to infer more realistic HR details by using additional physical quantities, such as LR velocities or vorticities. Besides improvements in the training process and in the generated outputs, these inputs offer means for artistic control as well. We additionally employ a physics-aware data augmentation step, which is crucial to avoid over-fitting and reduce memory requirements. In this way, our network learns to generate advected quantities with highly detailed, realistic, and temporally coherent features. Our method works instantaneously, using only a single time-step of LR fluid data. We demonstrate the abilities of our method using a variety of complex inputs and applications in two and three dimensions.

We additionally present best practices to set up a training pipeline for physics-based GANs. E.g., we found it particularly useful to have physics-aware data augmentation functionality in place. The large amounts of space-time data that arise in the context of many physics problems quickly bring typical hardware environments to their limits. As such, we found data augmentation crucial to avoid over-fitting. We also explored a variety of different variants for setting up the networks as well as training them, and we will evaluate them in terms of their capabilities to learn high-resolution physics functions below.

3.1 Network Architectures

As we target a super-resolution problem, our goal is not to generate an arbitrary high-resolution output, but one that corresponds to a low-resolution input, and hence we employ a *conditional* GAN conditioned on low-resolution input. As shown in Figure 3.1, our pipeline is built based on the GAN structure, containing a generator G, which can generate high-resolution data with low-resolution data as input, and a spatial discriminator D_s, which is trained to judge distinguish generated data from real data. Beyond those basic GAN components, we propose a novel temporal discriminator D_t to improve the temporal coherence of the results in an unsupervised manner. Below, we will demonstrate loss functions for each part of our pipeline in detail.

3.2 Loss functions

Based on a set of low-resolution inputs, with corresponding high-resolution references, our goal is to train a CNN that produces a temporally coherent, high-resolution solution with adversarial training. We will first very briefly summarize the basics of adversarial training, and then explain our extensions for temporal coherence and for results control.

3.2.1 Spatial Adversarial Loss

As discussed in Chapter 2, GANs consist of two models, which are trained in conjunction: the generator G and the discriminator D. Both will be realized as convolutional neural networks in our case. This discriminator takes the form of a simple binary classifier, which is trained in a supervised manner to reject *generated* data, i.e., it should return $D(G(x)) = 0$, and accept the *real* data with $D(y) = 1$. For training, the loss for the discriminator is thus given by a sigmoid cross-entropy for the two classes "generated" and "real":

$$\begin{aligned} \mathcal{L}_D(D,G) =& \mathbb{E}_{y\sim p_{\mathrm{y}}(y)}[-\log D(y)] + \mathbb{E}_{x\sim p_{\mathrm{x}}(x)}[-\log(1-D(G(x)))] \\ =& \mathbb{E}_m[-\log D(y_m)] + \mathbb{E}_n[-\log(1-D(G(x_n)))] \ , \end{aligned} \tag{3.1}$$

where n is the number of drawn inputs x, while m denotes the number of real data samples y. Here we use the notation $y \sim p_{\mathrm{y}}(y)$ for samples y being drawn from a corresponding probability data distribution p_{y}, which will, later on, be represented by our numerical simulation framework. The continuous distribution $\mathcal{L}_D(D,G)$ yields the average of discrete samples y_m and x_n in the second line of Equation 3.1. We will omit the $y \sim p_{\mathrm{y}}(y)$ and $x \sim p_{\mathrm{x}}(x)$ subscripts of the sigmoid cross-entropy, and m and n subscripts of $D(y_m)$ and $G(x_n)$, for clarity below.

In contrast to the discriminator, the generator is trained to "fool" the discriminator into accepting its samples and thus generate output that is close to the real data from y. In practice, this means that the generator is trained to drive the discriminator result for its outputs to one. Instead of directly using the negative discriminator loss, GANs typically use

$$\mathcal{L}_G(D,G) = \mathbb{E}_{x\sim p_{\mathrm{x}}(x)}[-\log(D(G(x)))] = \mathbb{E}_n[-\log(D(G(x)))] \tag{3.2}$$

as the loss function for the generator, in order to reduce diminishing gradient problems [10]. As D is realized as a NN, it is guaranteed to be sufficiently differentiable as a loss function for G. In practice, both discriminator and generator are trained in turns and will optimally reach an equilibrium state.

In our case, the input x now represents the low-resolution data set, and the discriminator of conditional GANs is provided with x in order to establish and ensure the correct relationship between input and output, i.e., we now have $D(x,y)$ and $D(x,G(x))$ [67].

3.2.2 Spatial L1 Loss

Furthermore, previous work [137] has shown that an additional L_1 loss term with a small weight can be added to the generator to ensure that its output stays close to the ground

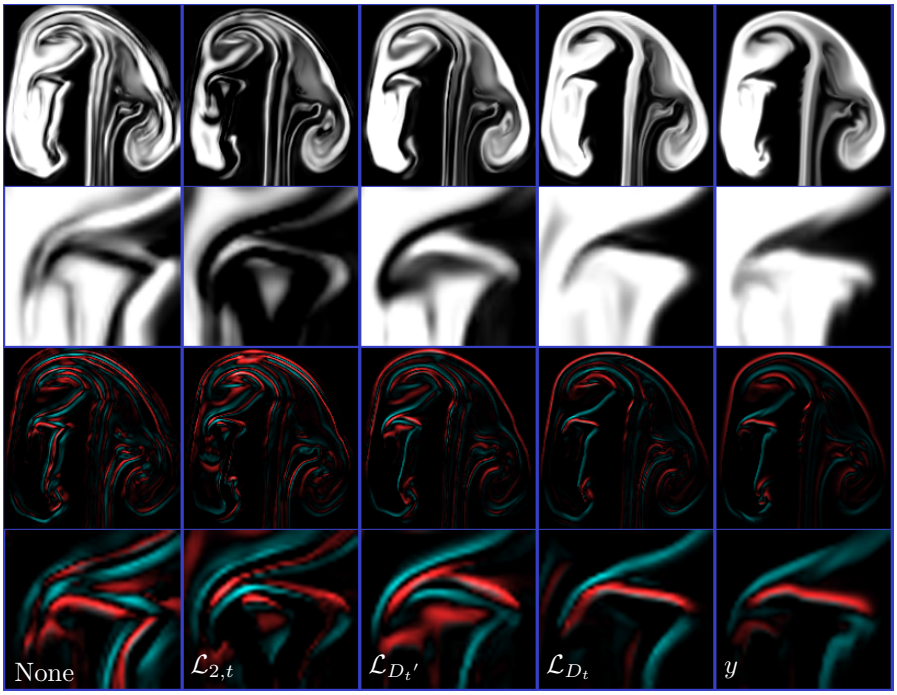


Figure 3.2: A comparison of different approaches for temporal coherence. The top two rows show the inferred densities, while the bottom two rows contain the time derivative of the frame content computed with a finite difference between frame t and $t+1$. Positive and negative values are color-coded with red and blue, respectively. From left to right: no temporal loss applied, $\mathcal{L}_{2,t}$ loss applied, $\mathcal{L}_{D_t'}$, i.e., applied without advection, $\mathcal{L}_{D_t}$ applied with advection (our full tempoGAN approach), and the ground-truth y. From left to right across the different versions, the derivatives become less jagged and less noisy, as well as more structured and narrow. This means the temporal coherence is improved, esp. for the result from our algorithm ($\mathcal{L}_{D_t}$).

truth y. This yields $\lambda_{L_1}\mathbb{E}_n \left\|G(x)-y\right\|_1$, where λ_{L_1} controls the strength of this term, and we use $\mathbb{E}$ for consistency to denote the expected value, in this discrete case being equivalent to an average.

3.2.3 Temporal Coherence

While the GAN process described so far is highly successful at generating highly detailed and realistic outputs for static frames, these details are particularly challenging in terms of their temporal coherence. Since both the generator and the discriminator work on every frame independently, subtle changes in input x can lead to outputs $G(x)$ with distinctly different details for higher spatial frequencies.

When the ground truth data y comes from a transport process, such as frame motion or flow motion, it typically exhibits a very high degree of temporal coherence, and a velocity field v_y exists for which $y^t = \mathcal{A}(y^{t-1}, v_y^{t-1})$. Here, we denote the advection operator (also

called warp or transport in other works) with $\mathcal{A}$, and we assume without loss of generality that the time step between frames t and $t-1$ is equal to one. Discrete time steps will be denoted by superscripts, i.e., for a function y of space and time $y^t = y(\mathbf{x}, t)$ denotes a full spatial sample at time t. Similarly, in order to solve the temporal coherence problem, the relationship $G(x^t) = \mathcal{A}(G(x^{t-1}), v^{t-1}_{G(x)})$ should hold, which assumes that we can compute a motion $v_{G(x)}$ based on the generator input x. While directly computing such a motion can be difficult and unnecessary for general GAN problems, we can make use of the ground truth data for y in our conditional setting. I.e., in the following, we will use a velocity reference v_y corresponding to the target y, and perform a spatial down-sampling to compute the velocity v_x for input x.

Equipped with v_x, one possibility to improve temporal coherence would be to add an L_2 loss term of the form:

$$\mathcal{L}_{2,t} = \|G(x^t) - \mathcal{A}(G(x^{t-1}), v_x^{t-1})\|_2^2 \tag{3.3}$$

We found that extending the forward-advection difference with backward-advection improves the results further, i.e., the following L_2 loss is clearly preferable over Equation 3.3:

$$\mathcal{L}_{2,t} = \|G(x^t) - \mathcal{A}(G(x^{t-1}), v_x^{t-1})\|_2^2 + \|G(x^t) - \mathcal{A}(G(x^{t+1}), -v_x^{t+1})\|_2^2 \tag{3.4}$$

, where we align the next frame at $t+1$ by advecting with $-v_x^{t+1}$.

While this $\mathcal{L}_{2,t}$ based loss improves temporal coherence, our tests show that its effect is relatively small. E.g., it can improve outlines but leads to clearly unsatisfactory results. One side effect of this loss term is that it can easily be minimized by simply reducing the values of $G(x)$. This is visible, e.g., in the second column of Figure 3.2, which contains noticeably less density than the other versions and the ground truth. However, we do not want to drive the generator towards darker outputs, but rather make it aware of how the data should change over time.

Instead of manually encoding the allowed temporal changes, we propose to use another discriminator D_t, that learns from the given data whose changes are admissible. In this way, the original spatial discriminator, which we will denote as $D_s(x, G(x))$ from now on, guarantees that our generator learns to generate realistic details, while the new temporal discriminator D_t mainly focuses on driving $G(x)$ towards solutions that match the temporal evolution of the ground-truth y.

Specifically, D_t takes three frames as input. We will denote such sets of three frames with a tilde in the following. As real data for the discriminator, the set $\widetilde{Y}_{\mathcal{A}}$ contains three consecutive and advected frames, thus $\widetilde{Y}_{\mathcal{A}} = \{\mathcal{A}(y^{t-1}, v_x^{t-1}),\ y^t,\ \mathcal{A}(y^{t+1}, -v_x^{t+1})\}$. The generated data set contains correspondingly advected samples from the generator: $\widetilde{G}_{\mathcal{A}}(\widetilde{X}) = \{\mathcal{A}(G(x^{t-1}), v_x^{t-1}),\ G(x^t),\ \mathcal{A}(G(x^{t+1}), -v_x^{t+1})\}$.

Similar to our spatial discriminator D_s, the temporal discriminator D_t is trained as a binary classifier on the two sets of data:

$$\mathcal{L}_{D_t}(D_t, G) = \mathbb{E}_m[-\log D_t\Big(\widetilde{Y}_{\mathcal{A}}\Big)] + \mathbb{E}_n[-\log(1 - D_t\Big(\widetilde{G}_{\mathcal{A}}\Big(\widetilde{X}\Big)\Big))], \tag{3.5}$$

where set $\widetilde{X}$ also contains three consecutive frames, i.e., $\widetilde{X} = \{x^{t-1},\ x^t, x^{t+1}\}$. Note that, unlike the spatial discriminator, D_t is not a conditional discriminator. It does not

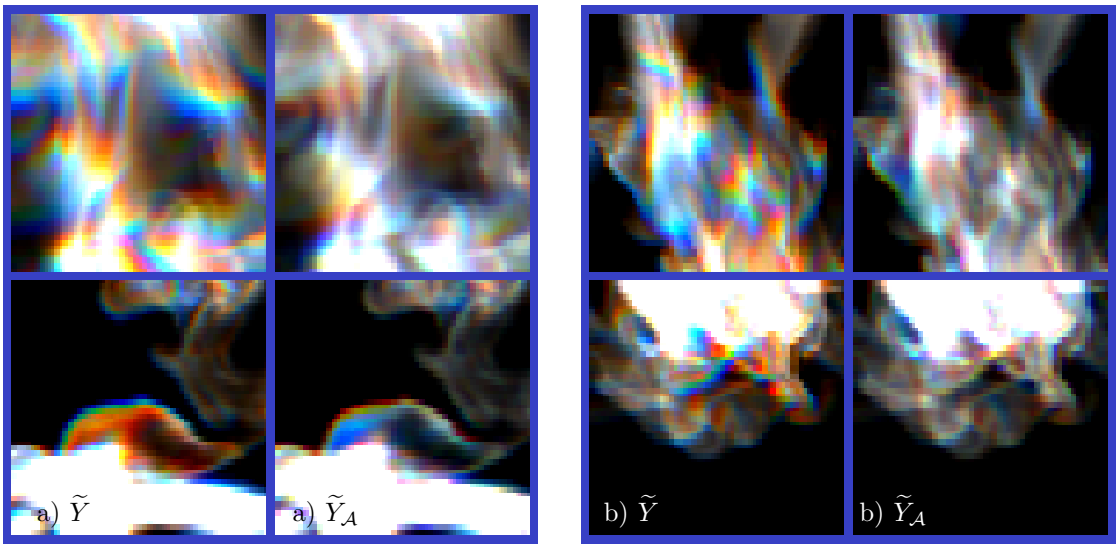


Figure 3.3: These images highlight data alignment due to advection. Three consecutive frames are encoded as R, G, B channels of a single image, thus, ideally a fully aligned image would only contain shades of grey. The two rows contain front and top views in the top and bottom row, respectively. We show two examples, a) and b). Each of them contains $\widetilde{Y}$ (left), and $\widetilde{Y}_{\mathcal{A}}$ (right). The RGB channels are the three input frames, t-1, t, and t+1. Compared with $\widetilde{Y}$, $\widetilde{Y}_{\mathcal{A}}$ is significantly less saturated, i.e., better aligned.

"see" the conditional input x, and thus D_t is forced to make its judgment purely based on the given sequence.

In Figure 3.2, we show a comparison of the different loss variants for improving temporal coherence. The first column is generated with only the spatial discriminator, i.e., provides a baseline for the improvements. The second column shows the result using the L_2-based temporal loss $\mathcal{L}_{2,t}$ from Equation 3.4, while the fourth column shows the result using D_t from Equation 3.5. The last column is the ground-truth data y. The first two rows show the generated density fields. While $\mathcal{L}_{2,t}$ reduces overall density content, the result with D_t is clearly closer to the ground truth. The bottom two rows show time derivatives of the densities for frames t and t+1. Again, the result from D_t and the ground-truth y match closely in terms of their time derivatives. The large and jagged values of the first two columns indicate the undesirable temporal changes produced by the regular GAN and the $\mathcal{L}_{2,t}$ loss.

In the third column of Figure 3.2, we show a simpler variant of our temporal discriminator. Here, we employ the discriminator without aligning the set of inputs with advection operations, i.e.,

$$\mathcal{L}_{D'_t}(D'_t, G) = \mathbb{E}_m[-\log D_t\Big(\widetilde{Y}\Big)] + \mathbb{E}_n[-\log(1 - D_t(\widetilde{G}\Big(\widetilde{X}\Big)))] \tag{3.6}$$

with $\widetilde{Y} = \{y^{t-1}, y^t, y^{t+1}\}$ and $\widetilde{G}\Big(\widetilde{X}\Big) = \{G(x^{t-1}), G(x^t), G(x^{t+1})\}$.

This version improves results compared to $\mathcal{L}_{2,t}$, but does not reach the level of quality of $\mathcal{L}_{D_t}$, as can be seen in Figure 3.2. Additionally, we found that $\mathcal{L}_{D_t}$ often exhibits a faster convergence during the training process. This is an indication that the underlying neural networks have difficulties aligning and comparing the data by themselves when using $\mathcal{L}_{D'_t}$. This intuition is illustrated in Figure 3.3, where we show example content of the regular data sets $\widetilde{Y}$ and the advected version $\widetilde{Y}_{\mathcal{A}}$ side by side. In this figure, the three chronological frames are visualized as red, green, and blue channels of the images. Thus, a pure gray-scale image would mean perfect alignment, while increasing visibility of individual colors indicates un-aligned features in the data. Figure 3.3 shows that, although not perfect, the advected one leads to clear improvements in terms of aligning the features of the data sets, despite only using the approximated coarse velocity fields v_x. Our experiments show that this alignment successfully improves the back-propagated gradients such that the generator learns to produce more coherent outputs. However, when no flow fields are available, $\mathcal{L}_{D'_t}$ still represents a better choice than the simpler $\mathcal{L}_{2,t}$ version. We see this as another indicator of the power of adversarial training models. It seems to be preferable to let a neural network learn and judge the specifics of a data set, instead of manually specifying metrics, as we have demonstrated for data sets of fluid flow motions above.

It is worth pointing out that our formulation for D_t in Equation 3.5 means that the advection step is an inherent part of the generator training process. While v_x can be pre-computed, it needs to be applied to the outputs of the generator during training. This in turn means that the advection needs to be tightly integrated into the training loop. The results discussed in the previous paragraph indicate that if this is done correctly, the loss gradients of the temporal discriminator are successfully passed through the advection steps to give the generator feedback such that it can improve its results. In the general case, advection is a non-linear function, the discrete approximation for which we have abbreviated with $\mathcal{A}(y^t, v_y^t)$ above. Given a known flow field v_y and time step, we can linearize this equation to yield a matrix $My = \mathcal{A}(y^t, v_y^t) = y^{t+1}$. E.g., for a first-order approximation, M would encode the Euler-step lookup of source positions and linear interpolation to compute the solution. While we have found first-order scheme (i.e., semi-Lagrangian advection) to work well, M could likewise encode higher-order methods for advection.

We have implemented this process as an advection layer in our network training, which computes the advection coefficients, and performs the matrix multiplication such that the discriminator receives the correct sets of inputs. When training the generator, the same code is used, and the underlying NN framework can easily compute the necessary derivatives. In this way, the generator actually receives three accumulated, and aligned gradients from the three input frames that were passed to D_t.

3.2.4 Loss in Feature Spaces

In order to further control the coupled, non-linear optimization process, the features of the underlying CNNs can be constrained. This is an important issue, as controlling the training process of GANs is known as a difficult problem. Here, we extend previous work

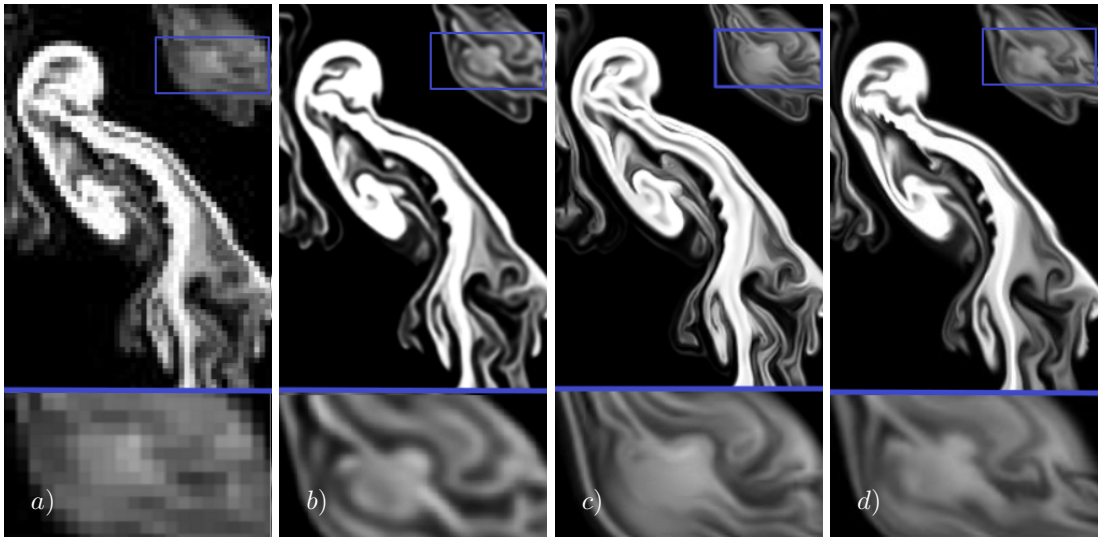


Figure 3.4: From left to right: a) a sample, low-resolution input, b) a CNN output with naive L_2 loss (no GAN training), c) our tempoGAN output, and d) the high-resolution reference. The L_2 version learns a smooth result without small scale details, while our output in (c) surpasses the detail of the reference in certain regions.

on feature space losses, which were shown to improve realism in natural images [138], and were also shown to help with mode collapse problems [139]. To achieve this goal, an L_2 loss over parts or the whole feature space of a neural network is introduced for the generator. I.e., the intermediate results of the generator network are constrained w.r.t. a set of intermediate reference data. While previous work typically makes use of manually selected layers of pre-trained networks, such as the VGG net, we propose to use features of the discriminator as constraints instead.

Thus, we incorporate a novel loss term of the form

$$\mathcal{L}_f = \mathbb{E}_{n,j} \lambda_f^j \left\| F^j(G(x)) - F^j(y) \right\|_2^2 \ , \tag{3.7}$$

where j is a layer in our discriminator network, and F^j denotes the activations of the corresponding layer. The factor λ_f^j is a weighting term, which can be adjusted on a per layer basis, as we will discuss in Sec. 3.5.2. It is particularly important in this case that we can employ the discriminator here, as no suitable, pre-trained networks are available for three-dimensional flow problems.

Interestingly, these weights yield different and realistic results both for positive as well as negative choices for the weights. For $\lambda_f > 0$ these loss terms effectively encourage minimization of the mean feature space distances of real and generated data sets, such that generated features resemble features of the reference. Surprisingly, we found that training runs with $\lambda_f < 0$ also yield excellent, and often slightly better results. As we are targeting conditional GANs, our networks are highly constrained by the inputs. Our explanation for this behavior is that a negative feature loss in this setting encourages the optimization to generate results that differ in terms of the features, but are still similar,

ideally indistinguishable, in terms of their final output. This is possible as we are not targeting a single ground-truth result, but rather, we give the generator the freedom to generate any result that best fits the collection of inputs it receives. From our experience, this loss term drives the generator towards realistic detail, an example of which can be seen in Figure 3.4. Note that due to the non-linear nature of the optimization, linearly changing λ_f yields to models with significant differences in the generated small scale features.

3.3 Training Details

3.3.1 Full algorithm

While the previous sections have explained the different parts of our final loss function, we summarize and discuss the combined loss in the following section. We will refer to our full algorithm as *tempoGAN*. The resulting optimization problem that is solved with NN training consists of three coupled non-linear sub-problems: the generator, the conditional spatial discriminator, and the un-conditional temporal discriminator. The generator has to effectively minimize both discriminator losses, additional feature space constraints, and a L_1 regularization term. Thus, the loss functions can be summarized as:

$$\begin{aligned}
\mathcal{L}_{D_t}(D_t, G) &= -\mathbb{E}_m[\log D_t(\widetilde{Y}_{\mathcal{A}})] - \mathbb{E}_n[\log\Big(1 - D_t\Big(\widetilde{G}_{\mathcal{A}}\Big(\widetilde{X}\Big)\Big)\Big)] \\
\mathcal{L}_{D_s}(D_s, G) &= -\mathbb{E}_m[\log D_s(x, y)] - \mathbb{E}_n[\log(1 - D_s(x, G(x)))] \\
\mathcal{L}_G(D_s, D_t, G) &= -\mathbb{E}_n[\log D_s(x, G(x))] - \mathbb{E}_n[\log D_t\Big(\widetilde{G}_{\mathcal{A}}\Big(\widetilde{X}\Big)\Big)] \\
&\quad + \mathbb{E}_{n,j}\lambda_f^j \left\| F^j(G(x)) - F^j(y) \right\|_2^2 + \lambda_{L_1}\mathbb{E}_n \left\| G(x) - y \right\|_1
\end{aligned} \tag{3.8}$$

Our generator has to effectively compete against two powerful adversaries, who, along the lines of "the enemy of my enemy is my friend", implicitly cooperate to expose the results of the generator. E.g., we have performed tests without D_s, only using D_t, and the resulting generator outputs were smooth in time, but clearly less detailed than when using both discriminators.

Among the loss terms of the generator, the L_1 term has a relatively minor role to stabilize the training by keeping the averaged output close to the target. However, due to the complex optimization problem, it is nonetheless helpful for successful training runs. The feature space loss, on the other hand, directly influences the generated features. In the adversarial setting, the discriminator most likely learns distinct features that only arise for the ground truth (positive features), or those that make it easy to identify generated versions, i.e., negative features that are only produced by the generator. Thus, while training, the generator will receive gradients to make it produce more features of the targets from $F(y)$, while the gradients from $F(G(x))$ will penalize the generation of recognizable negative features.

While positive values for λ_f reinforce this behavior, it is less clear why negative values can lead to even better results in certain cases. Our explanation for this behavior is that

the negative weights drive the generator towards distinct features that have to adhere to the positive and negative features detected by the discriminator, as explained above in Sec. 3.2.4, but at the same time differ from the average features in y. Thus, the generator cannot simply create different or no features, as the discriminator would easily detect this. Instead, it needs to develop features that are like the ones present in the outputs y, but don't correspond to the average features in $F(y)$, which, e.g., leads to the finely detailed outputs shown in Figure 3.4.

3.3.2 Training

We use the same modalities for all training runs: we employ the commonly used ADAM optimizer [1] with an initial learning rate of $2 \cdot 10^{-4}$ that decays to 1/20th for the second half of the training iterations. The number of training iterations is typically on the order of 10k. We use 20% of the data for testing and the remaining 80% for training. Our networks did not require any additional regularization such as dropout or weight decay. The training procedure is summarized again in Alg. 1. Due to the typically limited amount of GPU memory, especially for 3D data sets, we perform multiple training steps for each of the components. In Alg. 1, we use k_{D_s}, k_{D_t}, and k_G to denote the number training iterations for D_s, D_t, and G, respectively.

While the coupled non-linear optimization can yield different results even for runs with the same parameters due to the non-deterministic nature of parallelized operations, we found the results to be stable in terms of quality. In particular, we did not find it necessary to change the weights of the different discriminator loss terms. However, if desired, λ_f can be used to influence the learned details as described above. For training and running the trained networks, we use Nvidia GeForce GTX 1080 Ti GPUs (each with 11GB Ram) and Intel Core i7-6850K CPUs, while we used the *tensorflow* and *mantaflow* software frameworks for deep learning and fluid simulation implementations, respectively.

Algorithm 1 tempoGAN training algorithm

for number of training steps **do**
  for k_{D_s} **do**
    Compute data-augmented mini batch x, y
    Update D_s with $\nabla_{D_s}[\mathcal{L}_{D_s}(D_s, G)]$
  for k_{D_t} **do**
    Compute data-augmented mini batch $\widetilde{X}, \widetilde{Y}$
    Compute advected frames $\widetilde{Y}_{\mathcal{A}}$ and $\widetilde{G}_{\mathcal{A}}\left(\widetilde{X}\right)$
    Update D_t with $\nabla_{D_t}[\mathcal{L}_{D_t}(D_t, G)]$
  for k_G **do**
    Compute data-augmented mini batch $x, y, \widetilde{X}$
    Compute advected frames $\widetilde{G}_{\mathcal{A}}\left(\widetilde{X}\right)$
    Update G with $\nabla_G[\mathcal{L}_G(D_s, D_t, G)]$

[1] Parameterized with $\beta = 0.5$.

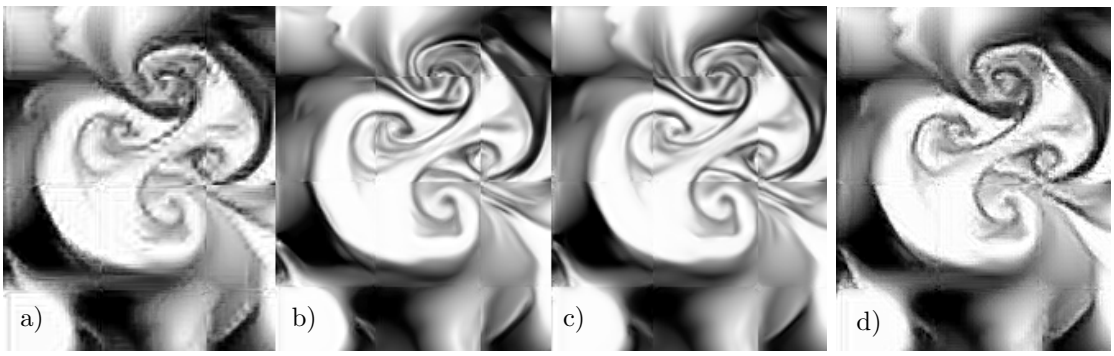


Figure 3.5: An illustration of different training results after 40k iterations with different input fields: a) ρ, b) $\rho + \mathbf{v}$, c) $\rho + \mathbf{v} + \mathbf{w}$, all with similar network sizes. Version d) with only ρ has 2x the number of weights. The seams in the images show the size of the training patches. Supplemental physical fields lead to clear improvements in b) and c), that even additional weights cannot compensate for.

3.3.3 Generator Inputs

At first sight, it might seem redundant and unnecessary to input flow velocity $\mathbf{v}$ and vorticity $\mathbf{w}$ in addition to the density ρ. After all, we are only interested in the final output density, and many works on GANs exist, which demonstrate that detailed images can be learned purely based on image content.

However, over the course of numerous training runs, we noticed that giving the networks additional information about the underlying physics significantly improves the convergence and quality of the inferred results. An example is shown in Figure 3.5. Here, we show how the training evolves for three networks with identical size, structure, and parameters, the only difference being the input fields. From left to right, the networks receive (ρ), $(\rho, \mathbf{v})$, and $(\rho, \mathbf{v}, \mathbf{w})$. Note that these fields are only given to the generator, while the discriminator always only receives (ρ) as input. The version with only density passed to the generator, $G(\rho)$, fails to reconstruct smooth and detailed outputs. Even after 40000 iterations, the results exhibit strong grid artifacts and lack detailed structures. In contrast, both versions with additional inputs start to yield higher quality outputs earlier during training. While adding $\mathbf{v}$ is crucial, the addition of $\mathbf{w}$ only yields subtle improvements (most apparent at the top of the images in Figure 3.5), which is why we will use $(\rho, \mathbf{v})$ to generate our final results below.

We believe that the insight that auxiliary fields help improve training and inference quality is a surprising and important one. The networks do not get any explicit guidance on how to use the additional information. However, it clearly not only learns to use this information but also benefits from having this supporting information about the underlying physics processes. While larger networks can potentially alleviate the quality problems of the density-only version, as illustrated in Figure 3.5 d), we believe it is highly preferable to instead construct and train smaller, physics-aware networks. This not only shortens training times and accelerates convergence, but also makes evaluating the trained model more efficient in the long run. The availability of physical inputs

turned out to be a crucial addition in order to successfully realize high-dimensional GAN outputs for space-time data, which we will demonstrate in Sec. 3.5.

3.4 Architecture and Training Data

While our loss function theoretically works with any realization of G, D_s and D_t, their specifics naturally have a significant impact on performance and the quality of the generated outputs. A variety of network architectures has been proposed for training generative models [57, 65, 140], and in the following, we will focus on pure convolutional networks for the generator, i.e., networks without any fully connected layers. A fully convolutional network has the advantage that the trained network can be applied to inputs of arbitrary sizes later on. We have experimented with a large variety of generator architectures, and while many simpler networks only yielded sub-optimal results, we have achieved high-quality results with generators based on the popular U-net [141, 68], as well as with residual networks (*res-nets*) [142]. The U-net concatenates activations from earlier layers to later layers (so-called *skip connections*) in order to allow the network to combine high- and low-level information, while the res-net processes the data using multiple *residual blocks*. Each of these residual blocks convolves the inputs without changing their spatial size, and the result of two convolutional layers is added to the original signal as a "residual" correction. In the following, we will focus on the latter architecture, as it gave slightly sharper results in our tests.

We found the discriminator architecture to be less crucial. As long as enough nonlinearity is introduced over the course of several hidden layers, and there are enough weights, changing the connectivity of the discriminator did not significantly influence the generated outputs. Thus, in the following, we will always use discriminators with four convolutional layers with leaky ReLU activations[2] followed by a fully connected layer to output the final score. As suggested by Odena et al. [143], we use the nearest-neighbor interpolation layers as the first two layers in our generator, instead of deconvolutional ones, and in the discriminator networks, the kernel size is divisible by the corresponding stride. An overview of the architecture of our neural networks is shown in Figure 3.6, while their details, such as layer configuration and activation functions, can be found in Appendix A.1.

3.4.1 Data Generation

We use a randomized smoke simulation setup to generate the desired number of training samples. For this we employ a standard fluids solver [18] with MacCormack advection and MiC-preconditioned CG solver. We typically generate around 20 simulations, with 120 frames of output per simulation. For each of these, we randomly initialize a certain number of smoke inflow regions, another set of velocity inflows, and a randomized buoyancy force. As inputs x, we use a down-sampled version of the simulation data sets, typically by a factor of 4, while the full resolution data is used as ground truth y. Note

[2] With a leaky tangent of 0.2 for the negative half space.

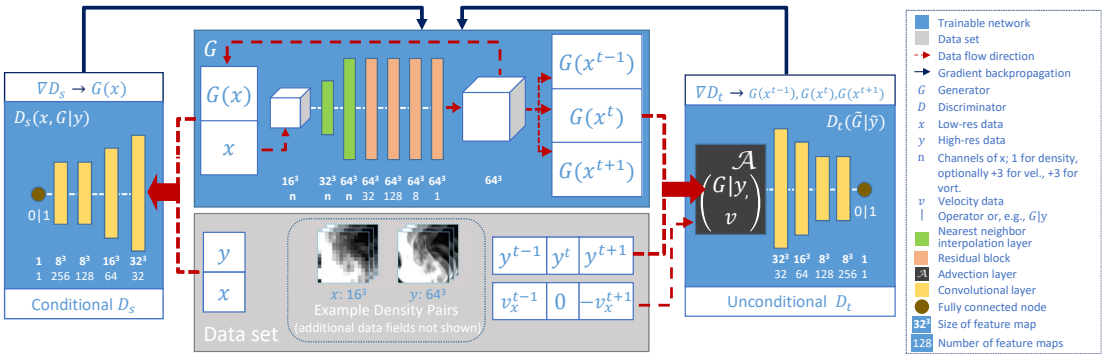


Figure 3.6: Here an overview of our tempoGAN architecture is shown. The three neural networks (blue boxes) are trained in conjunction. The data flow between them is highlighted by the red and black arrows. Note that x and y denote fluid data that contains velocity and/or vorticity fields, as well as density depending on the chosen architecture (see Sec. 3.3.3).

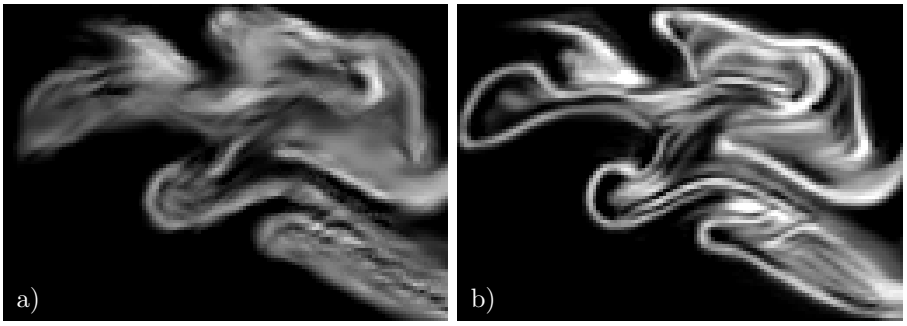


Figure 3.7: An identical GAN network trained with the same set of input data. While version a) did not use data augmentation, leading to blurry results with streak-like artifacts, version b), with data augmentation, produced sharp and detailed outputs.

that this setup is inherently *multi-modal*: for a single low-resolution configuration, an infinitely large number of correct high resolution exists. We do not explicitly sample the high-resolution solution space, but the down-sampling in conjunction with data augmentation leads to ambiguous low- and high-resolution pairs of input data. To prevent a large number of primarily empty samples, we discard inputs with an average smoke density of less than 0.02. Details of the parametrization can be found in Appendix A.2.

3.4.2 Data Augmentation

Data augmentation turned out to be an important component of our pipeline due to the high dimensionality of our data sets and the large amount of memory they require. Without sufficient enough training data, adversarial training yields undesirable results due to over-fitting. While data augmentation is common practice for natural images [144, 52], we describe several aspects below that play a role for physical data sets.

The augmentation process allows us to train networks having millions of weights with data sets that only contain a few hundred samples without over-fitting. At the same time, we can ensure that the trained networks respect the invariants of the underlying physical problems, which is crucial for the complex space-time data sets of flow fields that we are considering. E.g., we know from theory that solutions obey Galilean invariance, and we can make sure our networks are aware of this property not by providing large data sets, but instead by generating data with different inertial frames on the fly while training.

In order to minimize the necessary size of the training set without deteriorating the resulting quality, we generate modified data sets at training time. We focus on spatial transformations, which take the form of $\tilde{\mathbf{x}}(\mathbf{p}) = \mathbf{x}(A\mathbf{p})$, where $\mathbf{p}$ is a spatial position, and A denotes an 4×4 matrix. For applying augmentation, we distinguish three types of components of a data set:

- *passive*: these components can be transformed in a straightforward manner as described above. An example of passive components are the advected smoke fields ρ, shown in many of our examples.

- *directional*: the content of these components needs to be transformed in conjunction with the augmentation. A good example is velocity, whose directions need to be adjusted for rotations and flips, i.e., $\tilde{\mathbf{v}}(\mathbf{p}) = A_{3\times3}\mathbf{v}(A\mathbf{p})$, where $A_{3\times3}$ is the upper left 3×3 matrix of A.

- *derived*: finally, derived components would be invalid after applying augmentation, and thus need to be re-computed. A good example is physical quantities such as vorticity, which contain mixed derivatives that cannot be easily transformed into a new frame of reference. However, these quantities typically can be calculated anew from other fields after augmentation.

If the data set contains quantities that cannot be computed from other augmented fields, this, unfortunately, means that augmentation cannot be applied easily. However, we believe that a large class of typical physics data sets can in practice be augmented as described here.

For matrix A, we consider affine transformation matrices that contain combinations of randomized translations, uniform scaling, reflections, and rotations. Here, only those transformations are allowed that do not violate the physical model for the data set. While shearing and non-uniform scaling could easily be added, they violate the NS momentum equation and thus should not be used for flow data. We have used values in the range $[0.85, 1.15]$ for scaling, and rotations by $[-90, 90]$ degrees. We typically do not load derived components into memory for training, as they are re-computed after augmentation. Thus, they are computed on the fly for a training batch and discarded afterward.

The outputs of our simulations typically have significantly larger size than the input tiles that our networks receive. In this way, we have many choices for choosing offsets, in order to train the networks for shift invariance. This also aligns with our goal to train

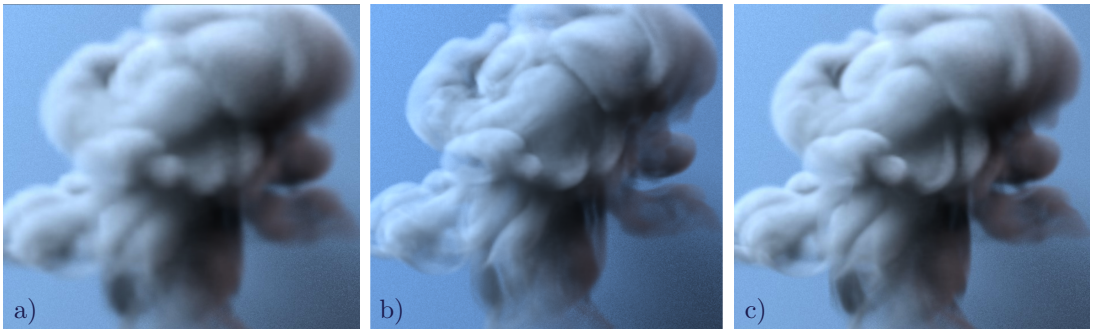


Figure 3.8: These images show our algorithm applied to a 3D volume. F.l.t.r.: a). a coarse input volume (down-sampled from the reference c, rendered with cubic up-sampling), b). our result, and c). the high-resolution reference. As in 2D, our trained model generates sharp features and detailed sheets that are at least on par with the reference.

a network that will later on work for arbitrarily sized inputs. We found it important to take special care at spatial boundaries of the tiles. While data could be extended by Dirichlet or periodic boundary conditions, it is important that the data set boundaries after augmentation do not lie outside the original data set. We enforce this by choosing suitable translations after applying the other transformations. This ensures that all data sets contain only valid content, and the network does not learn from potentially unphysical or unrepresentative data near boundaries. We also do not augment the time axis in the same way as the spatial axes. We found that the spatial transformations above applied to velocity fields give enough variance in terms of temporal changes. An example of the huge difference that data augmentation can make is shown in Figure 3.7. Here we compare two runs with the same amount of training data (160 frames of data), one with, the other one without data augmentation. While training a GAN directly with this data produces blurry results, the network converges to a final state with significantly sharper results with data augmentation. The possibility to successfully train networks with only a small amount of training data is what makes it possible to train networks for 3D+time data.

3.5 Results and Discussion

In the following, we will apply our method discussed so far to different data sets, and explore different application settings. Among others, we will discuss related topics such as art direction, training convergence, and performance.

3.5.1 3D Results

We have primarily used the 2D rising plume example in the previous sections to ensure the different variants can be compared easily. In Figure 3.8, we demonstrate that these results directly extend to 3D. We apply our method to a three-dimensional plume with

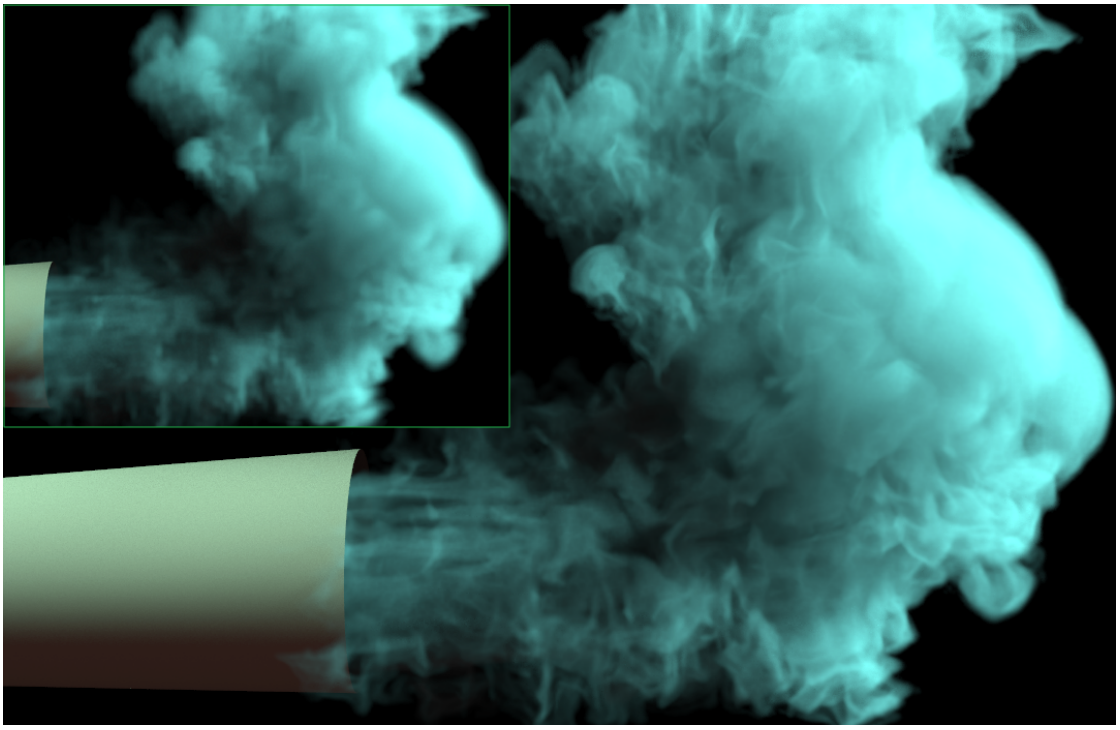


Figure 3.9: We apply our algorithm to a horizontal jet of smoke in this example. The inset shows the coarse input (rendered with cubic up-sampling), and the result of our algorithm. The diffuse streaks caused by procedural turbulence in the input (esp. near the inflow) are turned into detailed wisps of smoke by our algorithm.

resolution 64^3, which in this case was generated by down-sampling a 256^3 simulation such that we can compare our result to this reference solution. For this input data, the 256^3 output produced by our tempoGAN exhibits small-scale features that are at least as detailed as the ground truth reference. The temporal coherence is especially important in this setting, which is best seen in the accompanying video.

We also apply our trained 3D model to two different inputs with higher resolutions. In both cases, we use a regular simulation augmented with additional turbulence to generate an interesting set of inputs for our method. A first scene with $150 \times 100 \times 100$ is shown in Figure 3.9, where we generate a $600 \times 400 \times 400$ output with our method. The output closely resembles the input volumes but exhibits a large number of fine details. Note that our networks were only trained with down-sampled inputs, but our models generalize well to regular simulation inputs without re-sampling, as illustrated by this example.

Our method also has no problems with obstacles in the flow, as shown in Figure 3.10. This example has resolutions of $256 \times 180 \times 180$ and $1024 \times 720 \times 720$ for input and output volumes. The small-scale features closely adhere to the input flow around the obstacle. Although the obstacle is completely filled with densities towards the end of the simulation, there are no leaking artifacts as our method is applied independently to

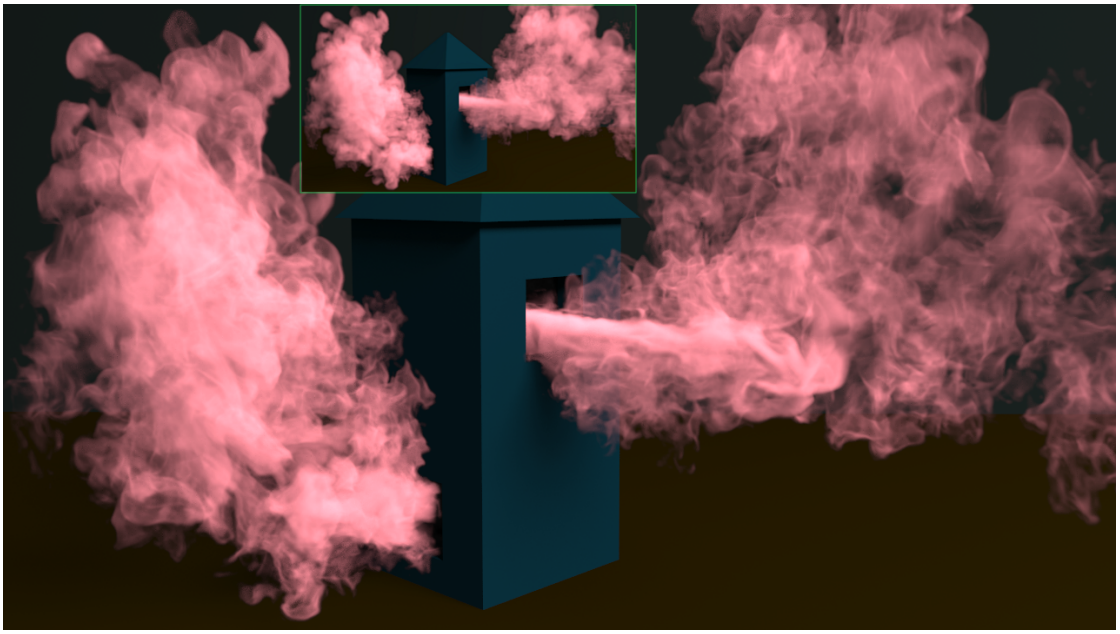


Figure 3.10: Our algorithm generated a high-resolution volume around an obstacle with a final resolution of $1024 \times 720 \times 720$. The inset shows the input volume. This scene is also shown in Figure 5.1 with a different visualization.

each input volume in the sequence. When showing the low-resolution input, we always employ cubic up-sampling, in order to not make the input look unnecessarily bad.

3.5.2 Fine Tuning Results

GANs have a reputation for being particularly hard to influence and control, and influencing the outcome of simulation results is an important topic for applications in computer graphics. In contrast to procedural methods, regular GAN models typically lack intuitive control knobs to influence the generated results. While we primarily rely on traditional guiding techniques to control the low-resolution input, our method offers different ways to adjust the details produced by our tempoGAN algorithm.

A first control knob for fine-tuning the results is to modify the data fields of the conditional inputs. As described in Sec. 3.3.3, our generator receives the velocity in addition to the density, and it internally builds tight relationships between the two. We can use these entangled inputs to control the features produced in the outputs. To achieve this, we modify the velocity components passed to the generator with various procedural functions. Figure 3.11 shows the results of the original input and several modified velocity examples and the resulting density configurations. We have also experimented with noise fields instead [67], but found that the trained networks completely ignored these fields. Instead, the strongly correlated velocity fields naturally provide much more meaningful input for our networks, and as a consequence provide means for influencing the results.

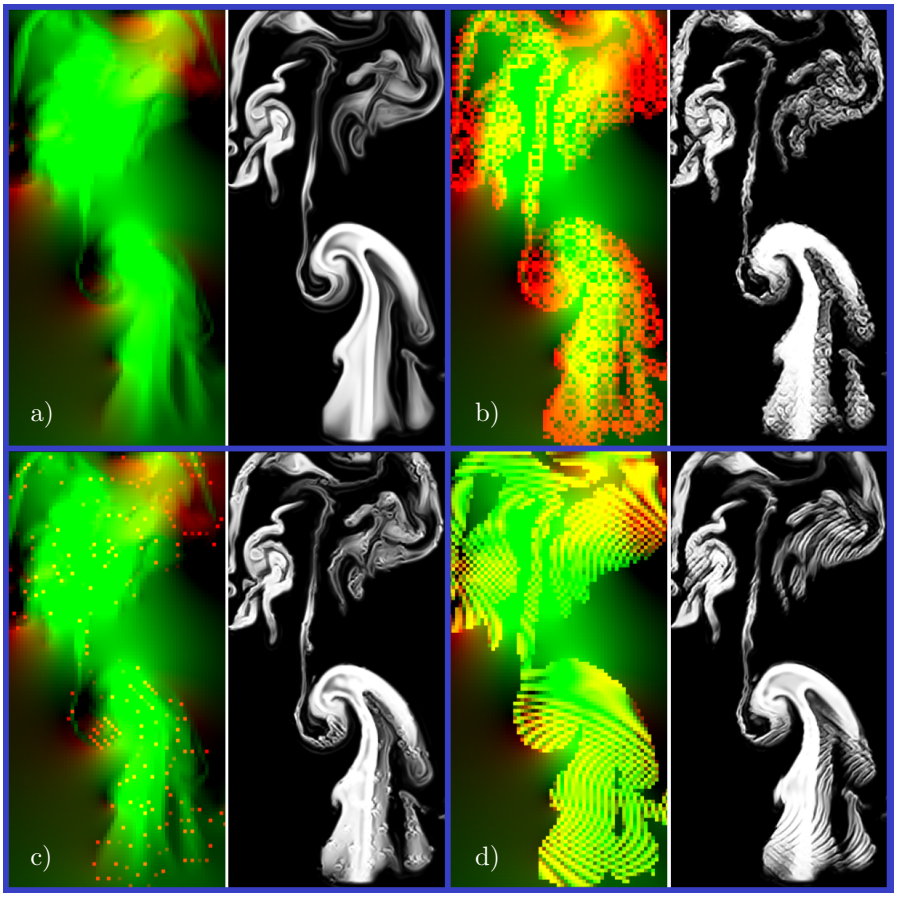


Figure 3.11: The red&green images on the left of each pair represent the modified velocity inputs, while the corresponding result is shown on the right. For reference, pair a) shows the unmodified input velocity and the regular output of our algorithm.

In addition, Figure 3.12 demonstrates that we can effectively suppress the generation of small scale details by setting all velocities to zero. Thus, the network learns a correlation between velocity magnitudes and amount of features. This is another indicator that the network learns to extract meaningful relationships from the data, as we expect turbulence and small-scale details to primarily form in regions with large velocities. Three-dimensional data can similarly be controlled, as illustrated in Figure 3.13.

In Sec. 3.2.4, we discussed the influence of the λ_f parameter for small scale features. For situations where we might not have additional channels such as the velocity above, we can use λ_f to globally let the network generate different features. However, as this only provides a uniform change that is encoded in the trained network, the resulting differences are more subtle than those from the velocity modifications above. Examples of different 2D and 3D outputs can be found in Figure 3.14 and Figure 3.15, respectively.

Figure 3.12: An illustration of how the entangled inputs of density and velocity can be used to fine-tune the results: on the left, the velocities were scaled up by a factor of 2, while the right-hand side was scaled by zero. The network has learned a relationship between detail and velocities, leading to reduced details in regions where the velocity was set to zero.

3.5.3 Additional Variants

In order to verify that our network can not only work with two- or three-dimensional data from a Navier-Stokes solver, we generated a more synthetic data set by applying strong wavelet turbulence to a 4× up-sampled input flow. We then trained our network with down-sampled inputs, i.e., giving it the task to learn the output of the wavelet turbulence algorithm. Note that a key difference here is that wavelet turbulence normally requires a full high-resolution advection over time, while our method infers high-resolution data sets purely based on low-resolution data from a single frame.

Our network successfully learns to generate structures similar to the wavelet turbulence outputs, shown in Figure 3.16. However, this data set turned out to be more difficult to learn than the original fluid simulation inputs. The training runs required two times more training data than the regular simulation runs, and we used a feature loss of $\lambda_f^{1,\dots,4} = 10^{-5}$. We assume that these more difficult training conditions are caused by the more chaotic nature of the procedural turbulence, and the less reasonable correlations between density and velocity inputs. Note that despite using more wavelet turbulence input data, it is still a comparatively small data set.

We additionally were curious about how well our network works when it is applied to a generated output, i.e., a recursive application. The result can be found in Figure 3.17, where we applied our network to its own output for an additional 2× up-sampling. Thus, in total this led to an 8× increase in resolution. While the output is plausible, and clearly contains even more fine features such as thin sheets, there is a tendency to amplify features generated during the first application.

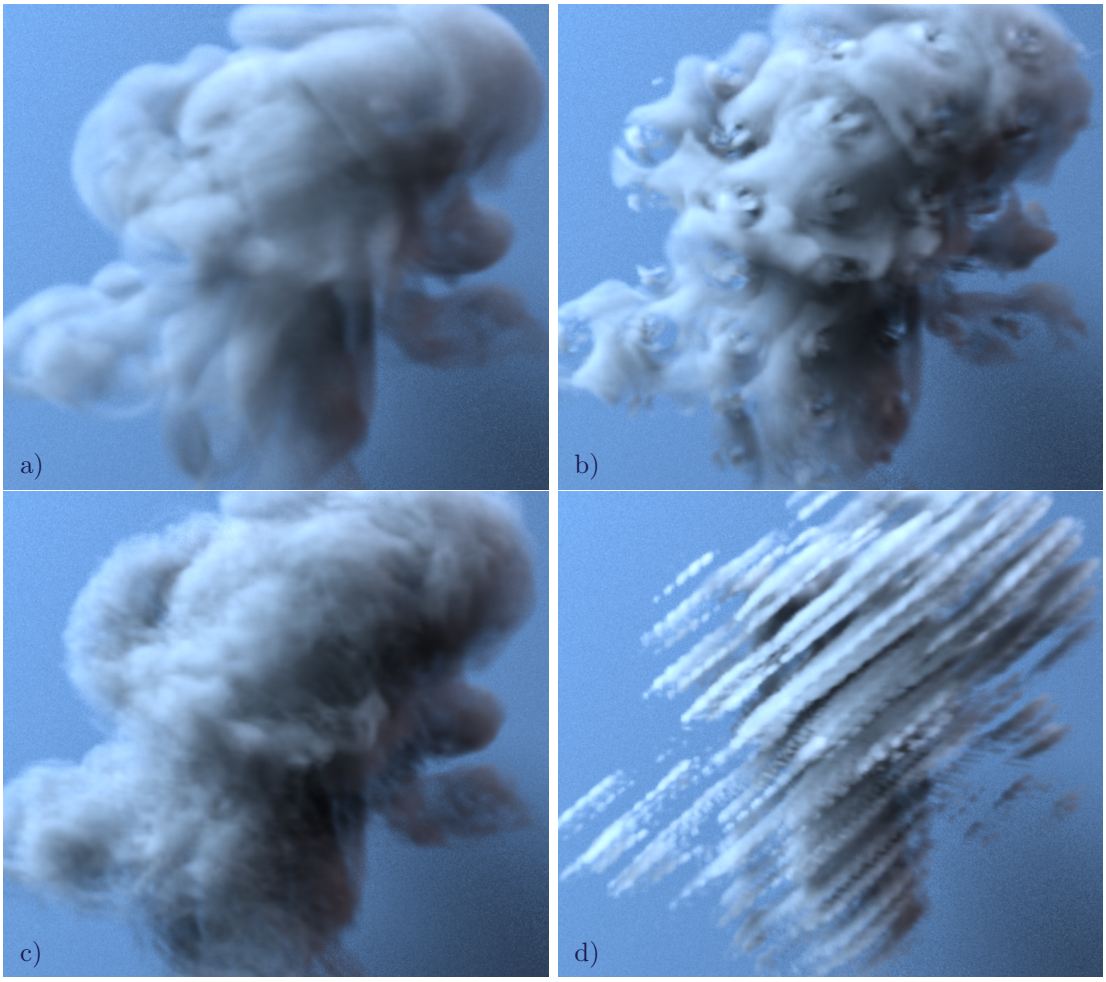


Figure 3.13: a) is the result of tempoGAN with velocity set to zero. The other three examples were generated with modified velocity inputs to achieve more stylized outputs.

3.5.4 Training Progress

With the training settings given in Appendix A.2, our training runs typically converged to stable solutions of around 1/2 for the discriminator outputs after sigmoid activation. While this by itself does not guarantee that a desirable solution was found, it at least indicates convergence towards one of the available local minima.

However, it is interesting how the discriminator loss changes in the presence of the temporal discriminator. Figure 3.18 shows several graphs of discriminator losses over the course of a full training run. Note that we show the final loss outputs from Equation 3.1 and Equation 3.5 here. A large value means the discriminator does "worse", i.e., it has more difficulty distinguishing real samples from the generated ones. Correspondingly, lower values mean it can separate them more successfully. In Figure 3.18a) it is visible that the spatial discriminator loss decreases when the temporal discriminator is intro-

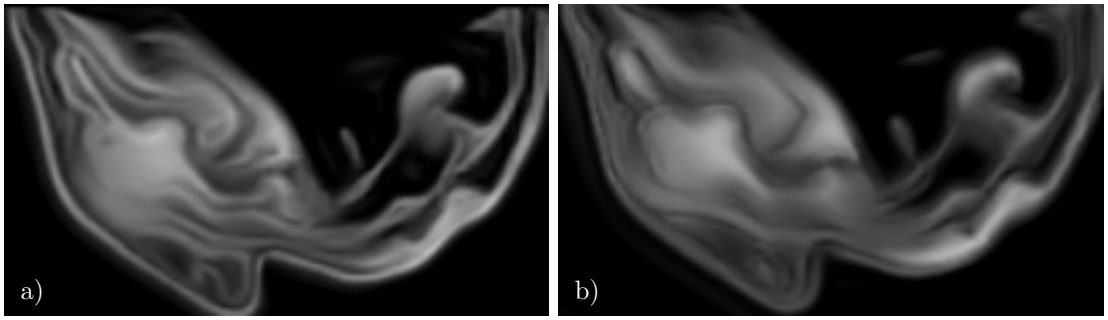


Figure 3.14: A comparison of training runs with different feature loss weights: a) $\lambda_f^{1,\dots,4} = -10^{-5}$, b) $\lambda_f^{1,4} = 1/3 \cdot 10^{-4}$, $\lambda_f^{2,3} = -1/3 \cdot 10^{-4}$.

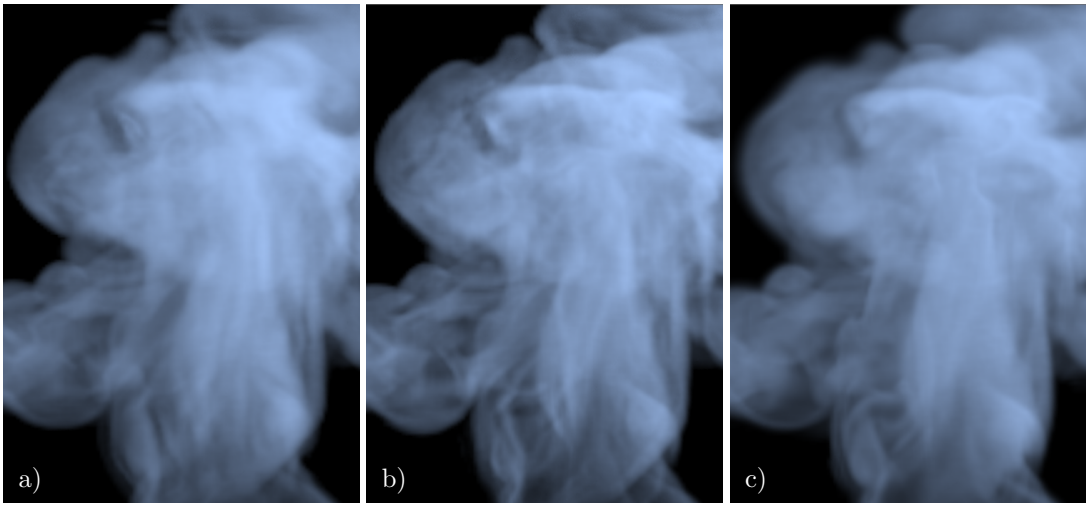


Figure 3.15: A comparison of training runs with different feature loss weights in 3D: a) with $\lambda_f^{1,\dots,4} = -1/3 \cdot 10^{-6}$, b) with $\lambda_f^{1} = 1/3 \cdot 10^{-6}$, $\lambda_f^{2,3,4} = -1/3 \cdot 10^{-6}$. The latter yields a sharpened result. Image c) shows the high resolution reference.

duced. Here the graph only shows the spatial discriminator loss, and the discriminator itself is unchanged when the second discriminator is introduced. The training run corresponding to the green line is trained with only a spatial discriminator, and for the orange line with both spatial and temporal discriminators. Our interpretation of the lower loss for the spatial discriminator network is that the existence of a temporal discriminator in the optimization prevents the generator from using the part of the solution space with detailed, but flickering outputs. Hence, the generator is driven to find a solution from the temporally coherent ones, and as a consequence has a harder time, which in turn makes the job easier for the spatial discriminator. This manifests itself as a lower loss for the spatial discriminator, i.e. the lower orange curve in Figure 3.18a).

Conversely, the existence of a spatial discriminator does not noticeably influence the temporal discriminator, as shown in Figure 3.18b). This is also intuitive, as the spatial discriminator does not influence temporal changes. We found that a generator trained

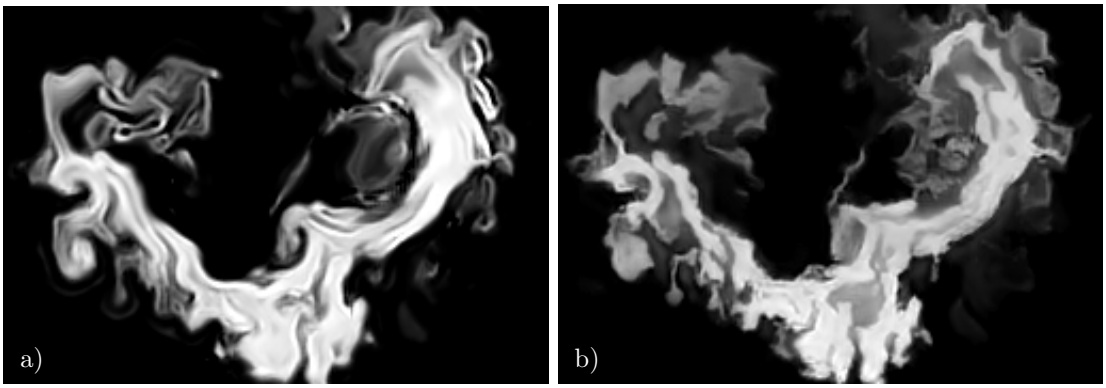


Figure 3.16: Our regular model a) and one trained with wavelet turbulence data b). In contrast to the model trained with real simulation data, the wavelet turbulence model produces flat regions with sharper swirls, mimicking the input data.

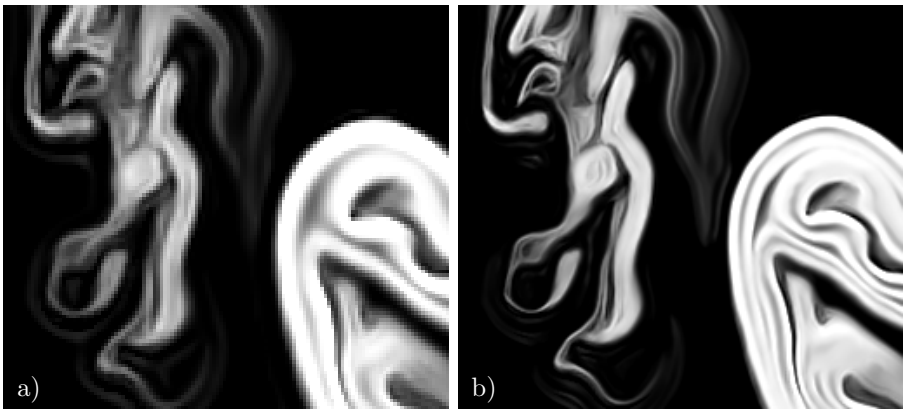


Figure 3.17: a) is the network output after a single application. b) is the network recursively applied to a) with a scaling factor of 2, resulting in a total increase of 8×.

only with D_t typically produces fewer details than a generator trained with both. In conjunction, our tests indicate that the two discriminators successfully influence different aspects of the solution space, as intended. Lastly, Figure 3.18c) shows that activating the negative feature loss from Sec. 3.2.4 makes the task for the generator slightly harder, resulting in a lowered spatial discriminator loss.

3.5.5 Performance

Training our two- and three-dimensional models is relatively expensive. Our full 2D runs typically take around 14 hours to complete (1 GPU), while the 3D runs took ca. 9 days using two GPUs. However, in practice, the state of the model after a quarter of this time is already indicative of the final performance. The remainder of the time is typically spent fine-tuning the network.

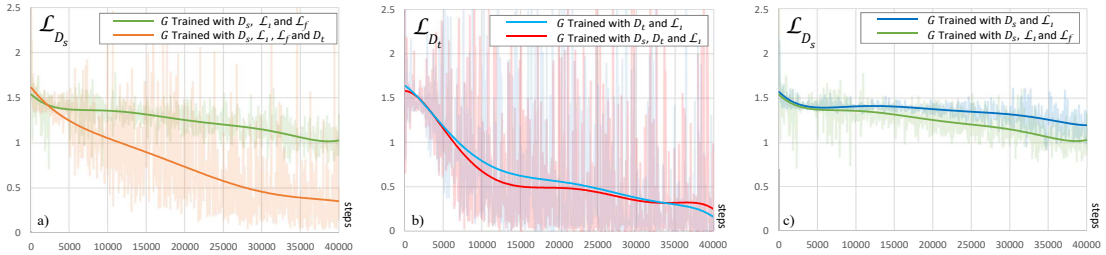


Figure 3.18: Several discriminator loss functions over the course of the 40k training iterations. a) D_s (spatial discriminator) loss is shown in green without D_t, and orange with D_t. b) Temporal discriminator loss in blue with only D_t, and in red for tempoGAN (i.e., with D_s, and feature loss). c) Spatial discriminator loss is shown in green with $\mathcal{L}_f$, and in dark blue without. For each graph, the dark lines show smoothed curves. The full data is shown in a lighter color in the background.

When using our trained network to generate high-resolution outputs in 3D, the limited memory of current GPUs poses a constraint on the volumes that can be processed at once, as the intermediate layers with their feature maps can take up significant amounts of memory. However, this does not pose a problem for generating larger final volumes, as we can subdivide the input volumes, and process them piece by piece. We generate tiles with a size of 136^3 on one GPU, with a corresponding input of size 34^3. Our 8 convolutional layers with a receptive field of 16 cells mean that up to four cells of an input could be influenced by a boundary. In practice, we found 3 input cells to be enough in terms of overlap. Generating a single 136^3 output took ca. 2.2 seconds on average. Thus, generating a 256^3 volume from a 64^3 input took 17.9s on average. Comparing the performance of our model with high-resolution simulations is inherently difficult, due to the substantially different implementations and hardware platforms (CPU vs. GPU). However, for the example of Figure 3.9 we estimate that fluid simulation at the full resolution would take ca. 31.5 minutes per frame of animation on average, while the evaluation of all volume tiles with our evaluation pipeline took ca. 3.9 minutes.

The cost for the trained model scales linearly with the number of cells in the volume, and in contrast to all previous methods for increasing the resolution of flow simulations, our method does not require any additional tracking information. It is also fully independent for all frames. Thus, our method could ideally be applied on the fly before rendering a volume, after which the high-resolution data could be discarded. Additionally, due to GPU memory restrictions, we currently evaluate our model in volumetric tiles with 3 cells of overlap for the input. This overlap can potentially be reduced further, and become unnecessary when enough memory is available to process the full input volume at once.

3.6 Limitations and Conclusions

Limitation One limitation of our approach is that the network encodes a fixed resolution difference for the generated details. While the initial up-sampling layers can be

stripped, and the network could thus be applied to inputs of any size, it will be interesting to explore different up-sampling factors beyond the factor of four which we have used throughout. With our current implementation, our method can also be slower than, e.g., calculating the advection for a high-resolution grid. However, a high-res advection would typically not lead to different dynamics than those contained in the input flow and require a sequential solve for the whole animation sequence. Our networks have so far also focused on buoyant smoke clouds. While obstacle interactions worked in our tests, we assume that networks trained for larger data sets and with other types of interactions could yield even better results.

Our three-dimensional networks needed a long time to train, circa nine days for our final model. Luckily, this is a one-time cost, and the network can be flexibly reused afterwards. However, if the synthesized small-scale features need to be fine-tuned, which we luckily did not find necessary for our work, the long runtimes could make this a difficult process. The feature loss weights clearly also are data-dependent, e.g., we used different settings for simulation and wavelet turbulence data. Here, it will be an interesting direction for future work to give the network additional inputs for fine-tuning the results beyond the velocity modifications discussed in Sec. 3.5.2.

Conclusion We have realized a first conditional GAN approach for four-dimensional data sets and we have demonstrated that it is possible to train generators that preserve temporal coherence using our novel time discriminator. The network architecture of this temporal discriminator, which ensures that the generator receives gradient information even for complex transport processes, makes it possible to robustly train networks for temporal evolutions. We have shown that this discriminator improves the generation of stable details as well as the learning process itself. At the same time, our fully convolutional networks can be applied to inputs of arbitrary size, and our approach provides basic means for the art direction of the generated outputs. We also found it very promising to see that our CNNs are able to benefit from coherent, physical information even in complex 3D settings, which led to reduced network sizes.

Overall, we believe that our contributions yield a robust and very general method for generative models of physics problems, and for super-resolution flows in particular. It will be highly interesting in future work to apply our tempoGAN to other physical problem settings, or even to non-physical data such as video streams.

4 Dataset Feature Extraction

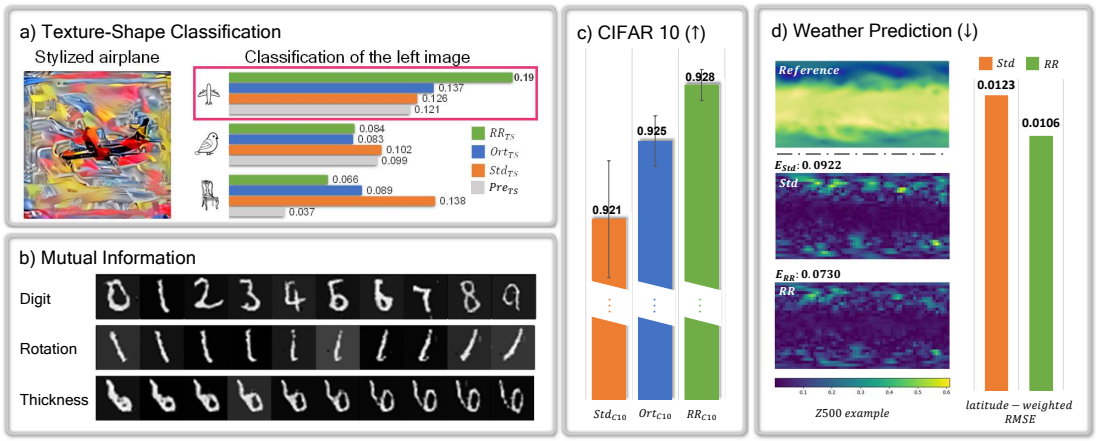


Figure 4.1: Our pretraining (denoted as RR) yields improvements for numerous applications: **a)** For difficult shape classification tasks, it outperforms existing approaches (Std_{TS}, Ort_{TS}, Pre_{TS}): the RR_{TS} model classifies the airplane shape with significantly higher confidence. **b)** Our approach establishes mutual information between input and output distributions. **c)** For CIFAR 10 classification with a Resnet110, RR_{C10} yields substantial practical improvements over the state-of-the-art. **d)** Learned weather forecasting likewise benefits from our pretraining, with RR yielding 13.7% improvements in terms of latitude-weighted RMSE for the ERA dataset [145]. Pressure is shown for 2019-08-09, 22:00 UTC, together with MAE for Std and RR models.

In the last chapter, we introduced tempoGAN, our temporally coherent SR algorithm for fluid flow. Once the tempoGAN model is trained, we can achieve efficient transformation from the LR fluid data to HR data. However, tempoGAN model requires to be retrained once the training data is changed, which is computationally expensive because of the large volume data size. This problem leads us to think about whether we can pretrain a base model, which can extract general and basic features of the fluid, then various models can be efficiently further trained based on this base model with specific fluid data. Similar ideas can be traced back to the image processing field. For example,

the VGG network is pretrained with imagenet data set [146], then features extracted by the VGG network can be efficiently reused in various tasks, such as image perceptual measuring [147], image stylization [148].

A common pretraining tool to extract features from the dataset is the autoencoder. While approaches such as greedy layer-wise autoencoder pretraining [149, 150, 151] paved the way for many fundamental concepts of today's methodologies in deep learning, the pressing need for pretraining neural networks has been diminished in recent years. An inherent problem is the lack of a global view: layer-wise pretraining is limited to adjusting individual layers one at a time. Thus, bottom layers that are optimized first cannot be adjusted to correct errors in higher layers [152]. In addition, numerous advances in regularization [153, 154, 155], network architectures [141, 156, 157], and improved optimization algorithms [158, 159, 160] have decreased the demand for layer-wise pretraining. Despite these advances, training deep neural networks that generalize well to a wide range of previously unseen tasks remains a fundamental challenge [161, 162, 163].

In this chapter, we develop an algorithm that reformulates autoencoder pretraining in a global way to arrive at a method that efficiently extracts general, dominant features from datasets. These features in turn improve performance for new tasks. Our approach is also inspired by techniques for orthogonalization [164, 165, 166]. Hence, we propose a modified variant that relies on a full reverse pass trained in conjunction with a given training task. A key insight is that there is no need for "greediness", i.e., layer-wise decompositions of the network structure, and it is additionally beneficial to take into account a specific problem domain at the time of pretraining. We establish links between singular value decomposition (SVD) and pretraining, and show how our approach yields an embedding of problem-aware dominant features in the weight matrices. An SVD can then be leveraged to conveniently gain insights about learned structures. Unlike orthogonalization techniques, we focus on embedding the dominant features of a dataset into the weights of a network. This is achieved via a *reverse pass* network. This reverse pass is generic, simple to construct, and directly relates to model performance, instead of, e.g., constraining the orthogonality of weights. Most importantly, we demonstrate that the proposed pretraining yields an improved performance for a variety of learning and transfer tasks. Our formulation incurs only a very moderate computational cost, which is very easy to integrate, and widely applicable.

The structure of our networks is influenced by invertible network architectures that have received significant attention in recent years [167, 168, 169]. However, these approaches rely heavily on specific network architectures. Instead of aiming for a bijective mapping that reproduces inputs, we strive for learning a general representation by constraining the network to represent an as-reversible-as-possible process for all *intermediate* layer activations. Thus, even for cases where a classifier can, e.g., rely on color for inference of an object type, the model is encouraged to learn a representation that can recover the input. Hence, not only the color of the input should be retrieved, but also, e.g., its shape, so that more dominant features of the input dataset are embedded into the networks. In contrast to most structures for invertible networks, our approach does not impose architectural restrictions. We demonstrate the benefits of our pretraining for

a variety of architectures, from fully connected layers to CNNs [52], over networks with batch normalization or dropout regularization, to GANs architectures [57].

Below, we will first give an overview of our formulation and its connection to singular values, before evaluating our model in the context of transfer learning. For a regular, i.e., a non-transfer task, the goal usually is to train a network that gives optimal performance for one specific goal. During a regular training run, the network naturally exploits any observed correlations between input and output distribution. An inherent difficulty in this setting is that typically no knowledge about the specifics of the new data and task domains is available when training the source model. Hence, it is common practice to target broad and difficult tasks hoping that this will result in features that are applicable in new domains [170, 171, 172]. Motivated by autoencoder pretraining, we instead leverage a pretraining approach that takes into account the data distribution of the inputs. We demonstrate the gains in accuracy for original and new tasks below for a wide range of applications, from image classification to data-driven weather forecasting.

4.1 Method Analysis

With state-of-the-art methods, there is no need for breaking down the training process into single layers. Hence, we consider approaches that target whole networks, and employ orthogonalization regularizers as a starting point [173]. Orthogonality constraints were shown to yield improved training performance in various settings [166], and for an n-layer network, they can be formulated as:

$$\mathcal{L}_{\text{ort}} = \sum_{m=1}^{n} \left\| M_m^T M_m - I \right\|_F^2 , \tag{4.1}$$

i.e., enforcing the transpose of the weight matrix $M_m \in \mathbb{R}^{s_m^{\text{out}} \times s_m^{\text{in}}}$ for all layers m to yield its inverse when being multiplied with the original matrix. I denotes the identity matrix with $I = (\mathbf{e}_m^1, ...\mathbf{e}_m^{s_m^{\text{in}}})$, $\mathbf{e}_m^j$ denoting the j_{th} column unit vector. Theoretically, $\mathcal{L}_{\text{ort}} = \mathbf{0}$ can not be perfectly fulfilled because of the information imbalance between inputs and outputs in most deep learning cases [174]. We will first analyze the influence of the loss function $\mathcal{L}_{\text{ort}}$ assuming that it can be fulfilled, before applying the analysis to our full pretraining method.

Minimizing Equation 4.1, i.e. $M_m^T M_m - I = \mathbf{0}$ is mathematically equivalent to:

$$M_m^T M_m \mathbf{e}_m^j - \mathbf{e}_m^j = \mathbf{0}, j = 1, 2, ..., s_m^{\text{in}}, m = 1, 2, ..., n, \tag{4.2}$$

with $rank(M_m^T M_m) = s_m^{\text{in}}$, and $\mathbf{e}_m^j$ as eigenvectors of $M_m^T M_m$ with eigenvalues of 1. This formulation highlights that Equation 4.2 does not depend on the training data, and instead only targets the content of M_m. Inspired by the classical unsupervised pretraining, we re-formulate the orthogonality constraint in a *data-driven* manner to take into account the set of inputs $\mathcal{D}_m$ for the current layer (either activation from a

previous layer or the training data $\mathcal{D}_1$), and instead minimize

$$\begin{aligned}\mathcal{L}_{\text{RR}} &= \sum_{m=1}^{n} \left\| M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i \right\|_2^2 \\ &= \sum_{m=1}^{n} \left\| (M_m^T M_m - I) \mathbf{d}_m^i \right\|_2^2,\end{aligned} \tag{4.3}$$

where $\mathbf{d}_m^i \in \mathcal{D}_m \subset \mathbb{R}^{s_m^{\text{in}}}$. Due to its reversible nature, we will denote our approach with an RR subscript in the following. In contrast to classical autoencoder pretraining, we are minimizing this loss jointly for all layers of a network, and while orthogonality only focuses on M_m, our formulation allows for minimizing the loss by extracting the dominant features of the input data.

Let q denotes the number of linearly independent entries in $\mathcal{D}_m$, i.e. its dimension, and t the size of the training data, i.e. $\mathcal{D}_m = t$, usually with $q < t$. For every single datum $\mathbf{d}_m^i, i = 1, 2, ..., t$, Equation 4.3 results in

$$M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i = \mathbf{0}, m = 1, 2, ..., n, \tag{4.4}$$

and hence $\mathbf{d}_m^i$ are eigenvectors of $M_m^T M_m$ with corresponding eigenvalues being 1. Thus, instead of the generic constraint $M_m^T M_m = I$ that is completely agnostic to the data at hand, the proposed formulation of Equation 4.4 is aware of the training data, which improves the generality of the learned representation, as we will demonstrate in detail below.

As by construction, $rank(M_m) = r \leqslant min(s_m^{\text{in}}, s_m^{\text{out}})$, the SVD of M_m yields:

$$\begin{aligned}&M_m = U_m \Sigma_m V_m^T, m = 1, 2, ..., n, \\ &\text{with} \begin{cases} U_m = (\mathbf{u}_m^1, \mathbf{u}_m^2, ..., \mathbf{u}_m^r, \mathbf{u}_m^{r+1}, ..., \mathbf{u}_m^{s_m^{\text{out}}}) \in \mathbb{R}^{s_m^{\text{out}} \times s_m^{\text{out}}}, \\ V_m = (\mathbf{v}_m^1, \mathbf{v}_m^2, ..., \mathbf{v}_m^r, \mathbf{v}_m^{r+1}, ..., \mathbf{v}_m^{s_m^{\text{in}}}) \in \mathbb{R}^{s_m^{\text{in}} \times s_m^{\text{in}}}, \end{cases}\end{aligned} \tag{4.5}$$

with left and right singular vectors in U_m and V_m, respectively, and Σ_m having square roots of the r eigenvalues of $M_m^T M_m$ on its diagonal. $\mathbf{u}_m^k$ and $\mathbf{v}_m^k (k = 1, ..., r)$ are the eigenvectors of $M_m M_m^T$ and $M_m^T M_m$, respectively [175]. Here, especially the right singular vectors in V_m^T are important, as they determine which structures of the input are processed by the transformation M_m. The original orthogonality constraint with Equation 4.2 yields r unit vectors $\mathbf{e}_m^j$ as the eigenvectors of $M_m^T M_m$. Hence, the influence of Equation 4.2 on V_m is completely independent of training data and learning objectives.

Next, we show that $\mathcal{L}_{\text{RR}}$ facilitates learning dominant features from a given dataset. For this, we consider an arbitrary basis for spanning the space of inputs $\mathcal{D}_m$ for layer m. Let $\mathcal{B}_m : \langle \mathbf{w}_m^1, ..., \mathbf{w}_m^q \rangle$ denote a set of q orthonormal basis vectors obtained via a Gram-Schmidt process, with $t \geqslant q \geqslant r$, and D_m denoting the matrix of the vectors in $\mathcal{B}_m$. As we show in more detail in the appendix, our constraint from Equation 4.4 requires eigenvectors of $M_m^T M_m$ to be $\mathbf{w}_m^i$, with V_m containing r orthogonal vectors $(\mathbf{v}_m^1, \mathbf{v}_m^2, ..., \mathbf{v}_m^r)$ from $\mathcal{D}_m$ and $(s_m^{\text{in}} - r)$ vectors from the null space of M.

We are especially interested in how M_m changes w.r.t. input in terms of D_m, i.e., we express $\mathcal{L}_{\text{RR}}$ in terms of D_m. By construction, each input $\mathbf{d}_m^i$ can be represented as a

linear combination via a vector of coefficients $\mathbf{c}_m^i$ that multiplies D_m so that $\mathbf{d}_m^i = D_m \mathbf{c}_m^i$. Since $M_m \mathbf{d}_m = U_m \Sigma_m V_m^T \mathbf{d}_m$, the loss $\mathcal{L}_{\text{RR}}$ of layer m can be rewritten as

$$\begin{aligned} \mathcal{L}_{\text{RR}_m} &= \left\| M_m^T M_m \mathbf{d}_m - \mathbf{d}_m \right\|_2^2 \\ &= \left\| V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{d}_m - \mathbf{d}_m \right\|_2^2 \\ &= \left\| V_m \Sigma_m^T \Sigma_m V_m^T D_m \mathbf{c}_m - D_m \mathbf{c}_m \right\|_2^2, m = 1, 2, ..., n, \end{aligned} \tag{4.6}$$

where we can assume that the coefficient vector $\mathbf{c}_m$ is accumulated over the training dataset size t via $\mathbf{c}_m = \sum_{i=1}^{t} \mathbf{c}_m^i$, since eventually every single datum in $\mathcal{D}_m$ will contribute to $\mathcal{L}_{\text{RR}_m}$. We can also rewrite Equation 4.6 with q components of D_m:

$$\mathcal{L}_{\text{RR}_m} = \left\| \sum_{h=1}^{q} V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h} - \mathbf{w}_m^h \mathbf{c}_{m_h} \right\|_2^2, m = 1, 2, ..., n. \tag{4.7}$$

This form of the loss highlights that minimizing $\mathcal{L}_{\text{RR}_m}$ requires an alignment of $\mathbf{w}_m^h \mathbf{c}_{m_h}$ and $V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h}$.

By construction, Σ_m contains the square roots of the eigenvalues of $M_m^T M_m$ as its diagonal entries. The matrix has rank $r = rank(M_m^T M_m)$, and since all eigenvalues are required to be 1 by Equation 4.4, the multiplication with Σ_m in Equation 4.7 effectively performs a selection of r column vectors from V_m. Hence, we can focus on the interaction between the basis vectors $\mathbf{w}_m$ and the r active column vectors of V_m:

$$\begin{aligned} & V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h} - \mathbf{w}_m^h \mathbf{c}_{m_h} \\ &= \mathbf{c}_{m_h} (V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h - \mathbf{w}_m^h) \\ &= \mathbf{c}_{m_h} (\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f - \mathbf{w}_m^h). \end{aligned} \tag{4.8}$$

As V_m is obtained via an SVD it contains r orthogonal eigenvectors of $M_m^T M_m$. Equation 4.4 requires $\mathbf{w}_m^1, ..., \mathbf{w}_m^q$ to be eigenvectors of $M_m^T M_m$, but since typically the dimension of the input dataset is much larger than the dimension of the weight matrix, i.e. $r \leq q$, in practice only r vectors from $\mathcal{B}_m$ can fulfill Equation 4.4. This means the vectors $\mathbf{v}_m^1, ..., \mathbf{v}_m^r$ in V_m are a subset of the orthonormal basis vectors $\mathcal{B}_m : \langle \mathbf{w}_m^1, ..., \mathbf{w}_m^q \rangle$ with $\left\| \mathbf{w}_m^h \right\|_2^2 = 1$. Then for any $\mathbf{w}_m^h$ we have

$$\begin{cases} (\mathbf{v}_m^f)^T \mathbf{w}_m^h = 1, & \text{if } \mathbf{v}_m^f = \mathbf{w}_m^h \\ (\mathbf{v}_m^f)^T \mathbf{w}_m^h = 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

Thus if V_m contains $\mathbf{w}_m^h$, we have

$$\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f = \mathbf{w}_m^h, \tag{4.10}$$

and we trivially fulfill the constraint

$$\mathbf{c}_{m_h}(\sum_{f=1}^{r}(\mathbf{v}_m^f)^T\mathbf{w}_m^h\mathbf{v}_m^f - \mathbf{w}_m^h) = \mathbf{0}. \tag{4.11}$$

However, due to r being smaller than q in practice, V_m typically can not include all vectors from $\mathcal{B}_m$. Thus, if V_m does not contain $\mathbf{w}_m^h$, we have $(\mathbf{v}_m^f)^T\mathbf{w}_m^h = 0$ for every vector $\mathbf{v}_m^f$ in V_m, which means

$$\sum_{f=1}^{r}(\mathbf{v}_m^f)^T\mathbf{w}_m^h\mathbf{v}_m^f = \mathbf{0}. \tag{4.12}$$

As a consequence, the constraint Equation 4.4 is only partially fulfilled:

$$\mathbf{c}_{m_h}(\sum_{f=1}^{r}(\mathbf{v}_m^f)^T\mathbf{w}_m^h\mathbf{v}_m^f - \mathbf{w}_m^h) = -\mathbf{c}_{m_h}\mathbf{w}_m^h\ . \tag{4.13}$$

As $\mathbf{w}_m^h$ has unit length, the factors $\mathbf{c}_m$ determine the contribution of a datum to the overall loss. A feature $\mathbf{w}_m^h$ that appears multiple times in the input data will have a correspondingly larger factor in $\mathbf{c}_m$ and hence will more strongly contribute to $\mathcal{L}_{\mathrm{RR}}$. The L^2 formulation of Equation 4.3 leads to the largest contributors being minimized most strongly, and hence the repeating features of the data, i.e., dominant features, need to be represented in V_m to minimize the loss. Interestingly, this argumentation holds when additional loss terms are present, e.g., a loss term for classification. In such a case, the factors $\mathbf{c}_m$ will be skewed towards those components that fulfill the additional loss terms, i.e. favor basis vectors $\mathbf{w}_m^h$ that contain information for about the loss terms. This, e.g., leads to clear digit structures being embedded in the weight matrices for the MNIST example below.

To summarize, V_m is driven towards containing r orthogonal vectors $\mathbf{w}_m^i$ that represent the most frequent features of the input data, i.e., the dominant features. Additionally, due to the column vectors of V_m being mutually orthogonal, M_m is encouraged to extract different features from the input. For the sake of being distinct and representative of the dataset, these features have the potential to be useful for new inference tasks. The feature vectors embedded in M_m can be extracted from the network weights in practical settings, as we will demonstrate below.

Realization in Neural Networks Calculating $M_m^TM_m$ is usually very expensive due to the dimensionality of M_m. Instead of building it explicitly, we constrain intermediate results to realize Equation 4.3 when training. Regular training typically starts with a chosen network structure and trains the model weights for a given task via a suitable loss function. Our approach fully retains this setup and adds a second pass that reverses the initial structure while reusing all weights and biases. For instance, for a typical fully connected layer in the forward pass with $\mathbf{d}_{m+1} = M_m\mathbf{d}_m + \mathbf{b}_m$, the reverse pass operation is given by $\mathbf{d}_m^{'} = M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m)$, where $\mathbf{d}_m^{'}$ denotes the reconstructed input.

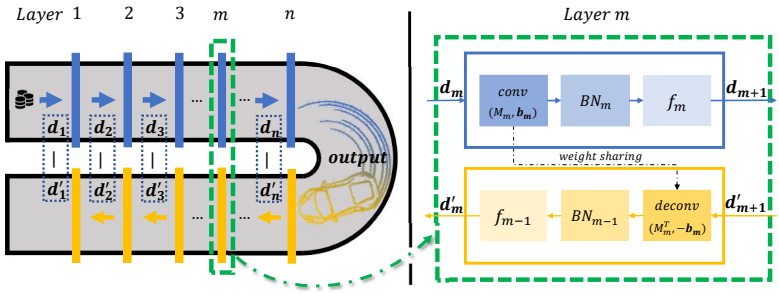


Figure 4.2: Left: An overview of the regular forward pass (blue) and the corresponding reverse pass (yellow). The right side illustrates how parameters are reused for a convolutional layer. *conv*/*deconv* denote convolution/deconvolutional operations. f_m and BN_m denote the activation function and batch normalization of layer m, respectively. Shared kernel and bias are represented by M_m and $\mathbf{b}_m$.

Our goal with the reverse pass is to transpose all operations of the forward pass to obtain identical intermediate activations between the layers with matching dimensionality. We can then constrain the intermediate results of each layer of the forward pass to match the results of the backward pass, as illustrated in Figure 4.2. While the construction of the reverse pass is straightforward for all standard operations, i.e., fully connected layers, convolutions, pooling, etc., slight adjustments are necessary for non-linear activation functions (AFs) and batch normalization (BN). It is crucial for our formulation that $\mathbf{d}_m$ and $\mathbf{d}'_m$ contain the same latent space content in terms of range and dimensionality, such that they can be compared in the loss. Hence, we use the BN parameters and the activation of layer $m-1$ from the forward pass for layer m in the reverse pass.

Unlike greedy layer-wise autoencoder pretraining, which trains each layer separately and only constrains $\mathbf{d}_1$ and $\mathbf{d}'_1$, we jointly train all layers and constrain all intermediate results. Due to the symmetric structure of the two passes, we can use a simple L^2 difference to drive the network towards aligning the results:

$$\mathcal{L}_{\text{RR}} = \sum_{m=1}^{n} \lambda_m \left\| \mathbf{d}_m - \mathbf{d}'_m \right\|_2^2. \tag{4.14}$$

Here $\mathbf{d}_m$ denotes the input of layer m in the forward pass and $\mathbf{d}'_m$ the output of layer m for the reverse pass. λ_m denotes a scaling factor for the loss of layer m, which, however, is typically constant in our tests across all layers. Note that with our notation, $\mathbf{d}_1$ and $\mathbf{d}'_1$ refer to the input data, and the reconstructed input, respectively.

Next, we show how this setup realizes the regularization from Equation 4.3. For clarity, we use a fully connected layer with bias. In a neural network with n hidden layers, the forward process for a layer m is given by $\mathbf{d}_{m+1} = M_m \mathbf{d}_m + \mathbf{b}_m$, with $\mathbf{d}_1$ and $\mathbf{d}_{n+1}$ denoting in- and output, respectively. All neural networks can be classified according to whether the full reverse pass can be built from the output to input, and we also classify our pretraining as full network pretraining and localized pretraining in implementation.

Full Network Pretraining: For networks where a unique path from output to input exists, we build a reverse pass network with transposed operations starting with the final output where $\mathbf{d}_{n+1} = \mathbf{d}'_{n+1}$, and the intermediate results $\mathbf{d}'_{m+1}$:

$$\mathbf{d}'_m = M_m^T(\mathbf{d}'_{m+1} - \mathbf{b}_m), m = 1, 2, ..., n, \tag{4.15}$$

where the reverse pass activation $\mathbf{d}'_m$ depends on $\mathbf{d}_{m+1}{}'$, this formulation yields a full reverse pass from output to input, which we use for most training runs below. Here we analyze the influence of Equation 4.14 during training by assuming $\mathcal{L}_{\text{RR}} = 0$ during the minimization. We then obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e. $\mathbf{d}'_{m+1} = \mathbf{d}_{m+1}$ holds. In this case

$$\begin{aligned} \mathbf{d}'_m &= M_m^T(\mathbf{d}'_{m+1} - \mathbf{b}_m) \\ &= M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m) = M_m^T M_m \mathbf{d}_m, m = 1, 2, ..., n, \end{aligned} \tag{4.16}$$

which means that Equation 4.14 is consistent with Equation 4.3.

Localized Pretraining: For architectures that have a reverse path that is not unique, e.g., in the presence of additive residual connections, we cannot uniquely determine the b, c in $a = b + c$ given only a. In such cases, we use a local formulation, and d_{m+1} is used as input of the reverse path of layer m directly. In this case Equation 4.15 can be written as:

$$\mathbf{d}'_m = M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m), m = 1, 2, ..., n, \tag{4.17}$$

which effectively employs $\mathbf{d}_{m+1}$ for jointly constraining all intermediate activations in the reverse pass. Moreover, it is consistent with Equation 4.3.

In summary, Equation 4.14 will drive the network towards a state that is as-invertible-as-possible for the given input dataset. Comparing the full network pretraining and localized pretraining, the full network pretraining establishes a stronger relationship among the loss terms of different layers, and allows earlier layers to decrease the accumulated loss of later layers. Localized pretraining, on the other hand, is even valid for cases where the reverse path from output to input is not unique.

Up to now, the discussion focused on simplified neural networks with convolutional operations, which are crucial for feature extraction, but without AFs or extensions such as BN, which are applied to increase model non-linearity. While we leave a more detailed theoretical analysis of these extensions for future work, we apply these non-linear extensions for all of our tests in sections 4.2 and 4.4. Thus, our experiments demonstrate that our method works in conjunction with BN and AFs. They show consistently show that the inherent properties of our pretraining remain valid: even in the non-linear setting our approach successfully extracts dominant structures and yields improved generalization.

In Sec. 4.3, we will give details on how to ensure that the latent space content for forward and reverse pass is aligned such that differences can be minimized, and we give practical examples of full and localized pretraining architectures.

To summarize, we realize the loss formulation of Equation 4.14 to minimize Equation 4.3without explicitly having to construct $M_m^T M_m$. Following the notation above, we will refer to networks trained with the added reverse structure and the additional loss terms as *RR* variants. We consider two variants for the reverse pass: a local pretraining Equation 4.17 using the datum $\mathbf{d}_{m+1}$ of a given layer, and a full version via

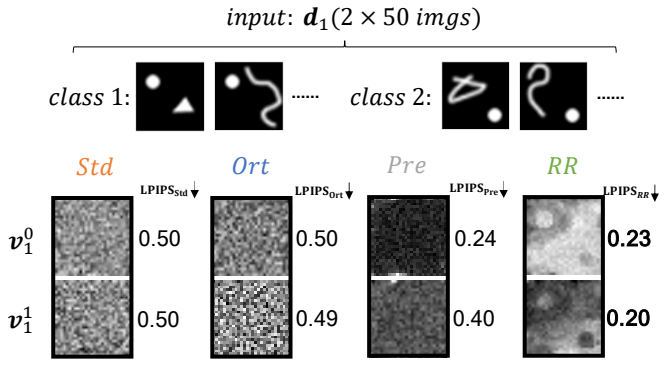


Figure 4.3: Column vectors of V_m for different trained models Std, Ort, Pre and RR for peaks. Input features clearly are successfully embedded in the weights of RR, as confirmed by the LPIPS scores.

Equation 4.15 which uses $\mathbf{d}'_{m+1}$ incoming from the next layer during the reverse pass. More details about the architecture will be illustrated in Sec. 4.3.

Embedding Singular Values In the following, we evaluate networks trained with different methodologies. We distinguish our pretraining approach RR (in green), regular autoencoder pretraining Pre (in grey), and and orthogonality constraints Ort (in blue). In addition, Std denotes a regular training run (in orange color in graphs below), i.e., models trained without autoencoder pretraining, orthogonality regularization or our proposed method. Besides, a subscript will denote the task variant the model was trained for, such as Std_T for task T. While we typically use all layers of a network in the constraints, a reduced variant that we compare below only applies the constraint for the input data, i.e., m=1. A network trained with this variant, denoted by RR^1_A, is effectively trained to only reconstruct the input. It contains no constraints for the inner activations and layers of the network. For the Ort models, we use the Spectral Restricted Isometry Property algorithm [166].

We verify that the column vectors of V_m of models from RR training contain the dominant features of the input with the help of a classification test, employing a single fully connected layer, i.e. $\mathbf{d}_2 = M_1\mathbf{d}_1$, with BN and activation. To quantify this similarity, we compute an LPIPS distance [147] between v_m^i and the training data (lower values being better). We employ a training dataset constructed from two dominant classes (a peak in the top left, and bottom right quadrant, respectively), augmented with noise in the form of random scribbles, as shown in Figure 4.3. Based on the analysis above, we expect the RR training to extract the two dominant peaks during training. The LPIPS measurements confirm our SVD argumentation above, with average scores of 0.217 ± 0.022 for RR, 0.319 ± 0.114 for Pre, 0.495 ± 0.006 for Ort, and 0.500 ± 0.002 for Std. I.e., the RR model fares significantly better than the others. At the same time, the peaks are clearly visible for RR models, while the other models fail to extract structures that resemble the input. Thus, by training with the full network and the original training objective, our

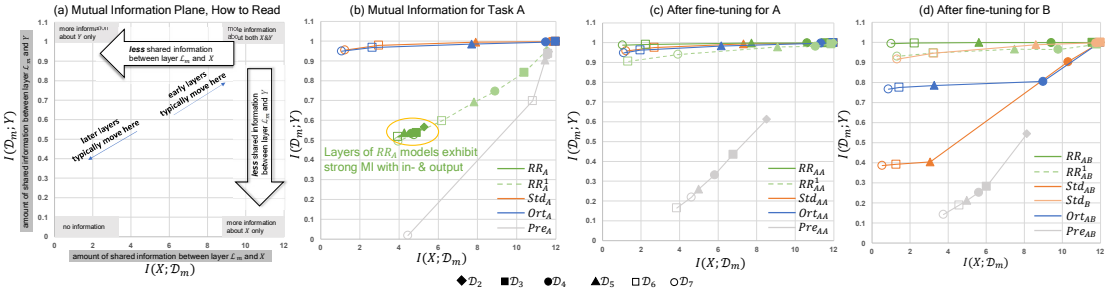


Figure 4.4: MI planes for different models: **a)** Visual overview of the contents. **b)** Plane for task A. Points on each line correspond to layers of one type of model. All points of RR_A, are located in the center of the graph, while Std_A and Ort_A, exhibit large $I(\mathcal{D}_m;Y)$, i.e., specialize on the output. Pre_A strongly focuses on reconstructing the input with high $I(X;\mathcal{D}_m)$ for early layers. **c, d)** After fine-tuning for A/B. The last layer $\mathcal{D}_7$ of RR_AA and RR_AB successfully builds the strongest relationship with Y, yielding the highest accuracy.

pretraining yields structures that are interpretable and be inspected by humans. More details about the tests are illustrated in Appendix B.

The results above experimentally confirm our formulation of the RR loss and its ability to extract dominant and generalizing structures from the training data. In addition, they give the first indication that this still holds when non-linear components such as AFs are present. Next, we will focus on quantified metrics and turn to measurements in terms of mutual information to illustrate the behavior of our pretraining for deeper networks.

4.2 Mutual Information

As our approach hinges on the introduction of the reverse pass, we will show that it succeeds in terms of establishing mutual information (MI) between the input and the constrained intermediates inside a network. More formally, MI $I(X;Y)$ of random variables X and Y measures how different the joint distribution of X and Y is w.r.t. the product of their marginal distributions, i.e., the Kullback-Leibler divergence $I(X;Y) = D_{KL}[P_{(X,Y)}||P_X P_Y]$. [174] proposed *MI plane* to analyze trained models, which show the MI between the input X and activations of a layer $\mathcal{D}_m$, i.e., $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$, i.e., MI of layer $\mathcal{D}_m$ with output Y. These two quantities indicate how much information about the in- and output distributions are retained at each layer, and we use them to show to which extent our pretraining succeeds at incorporating information about the inputs throughout training. Details of the settings can be found in Appendix B.

The following tests employ networks with six fully connected layers and non-linear AFs, with the objective to learn the mapping from 12 binary inputs to 2 binary output digits [176], with results accumulated over five runs. We compare the versions Std_A, Pre_A, Ort_A, RR_A, and a variant of the latter: RR_A^1, i.e. a version where only the input $\mathbf{d}_1$ is constrained to be reconstructed. While Figure 4.4 (a) visually summarizes the

content of the MI planes, the graph in Figure 4.4 (b) highlights that training with the RR loss correlates input and output distributions across all layers: the cluster of green points in the center of the graph shows that all layers contain balanced MI between in- as well as output and the activations of each layer. RR^1_A fares slightly worse, while Std_A and Ort_A almost exclusively focus on the output with $I(\mathcal{D}_m; Y)$ being close to one. Pre_A instead only focuses on reconstructing inputs. Thus, the early layers cluster in the right-top corner, while the last layer $I(\mathcal{D}_7; Y)$ fails to align with the outputs. Once we continue fine-tuning these models without regularization, the MI naturally shifts towards the output, as shown in Figure 4.4 (c). Here, RR_AA outperforms the other models in terms of the final performance. Likewise, RR_AB performs best for a transfer task B with switched output digits, as shown in graph (d). The final performance for both tasks across all runs is summarized in Figure 4.5. The graph shows that the proposed pretraining succeeds in robustly establishing mutual information between inputs and targets across a full network while extracting reusable features. The non-linearity of the underlying network architectures does not impede the performance of the RR models.

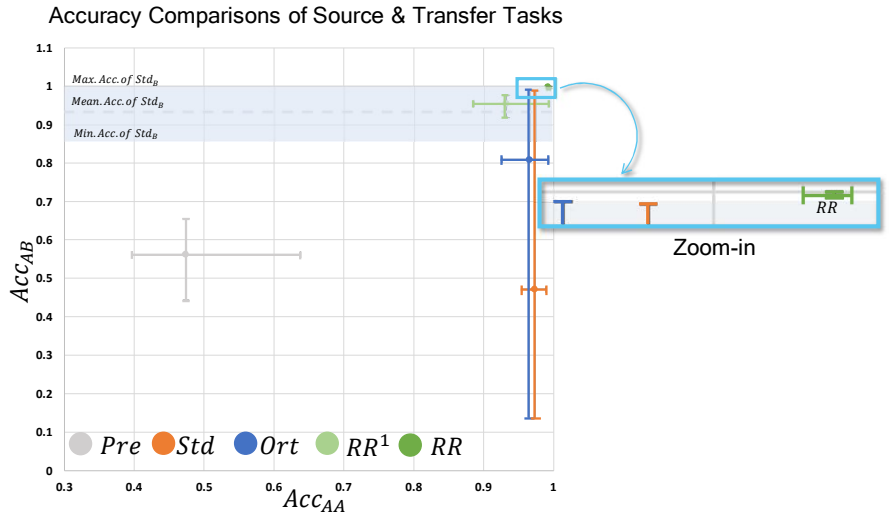


Figure 4.5: Performance for MI source and transfer tasks for the models of Figure 4.4. Due to the large standard deviation of Ort, we show min/max value ranges. The dashed gray line and region show the baseline accuracy for Std_B. The top-left inset highlights the stability of the high accuracy results from RR training.

MI has received attention recently as a learning objective, e.g., in the form of the InfoGAN approach [177] for learning disentangled and interpretable latent representations. While MI is typically challenging to assess and estimate [178], the results above show that our approach provides a straightforward and robust way for including it as a learning objective. In this way, we can easily, e.g., reproduce the disentangling results from [177] without explicitly calculating mutual information, which are shown in Figure 5.1(c) and Figure 4.6. A generative model with our pretraining extracts intuitive latent dimensions for the different digits, line thickness, and orientation without

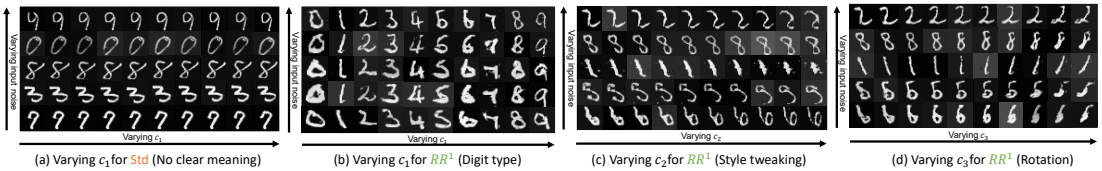


Figure 4.6: Additional results for the disentangled representations with the MNIST data: For every row in the figures, we vary the corresponding latent code (left to right), while keeping all other inputs constant. Different rows indicate a different random noise input. For example, in **b)** every column contains five results which are generated with different noise samples, but the same latent codes $c_{1\sim3}$. In every row, 10 results are generated with 10 different values of c_1, which correspond to one digit each for **b)**. **a)** For a regular training (Std), no clear correspondence between c_1 and the outputs are apparent (similarly for $c_{2,3}$). **c)** Different c_2 values result in a tweaked style, while c_3 controls the orientation of the digit, as shown in **d)**. Thus, in contrast to Std, the pretrained model learns a meaningful, disentangled representation.

any additional modifications to the loss function. The joint training of the full network with the proposed reverse structure, including non-linearities and normalization, yields a natural and intuitive decomposition.

4.3 Pretraining Network Architectures

While the proposed pretraining is significantly easier to integrate into training pipelines than classic autoencoder pretraining, there are subtleties w.r.t. the order of the operations in the reverse pass that we clarify with examples in this section. To specify NN architectures, we use the following notation: $C(k, l, q)$, and $D(k, l, q)$ denote convolutional and deconvolutional operations, respectively, while fully connected layers are denoted with $F(l)$, where k, l, q denote kernel size, output channels and stride size, respectively. The bias of a CNN layer is denoted with b. $I/O(z)$ denote *input*/*output*, their dimensionality is given by z. I_r denotes the input of the reverse pass network. *tanh*, *relu*, *lrelu* denote hyperbolic tangent, ReLU, and leaky ReLU activation functions (AFs), where we typically use a leaky tangent of 0.2 for the negative half-space. UP, MP and BN denote $2\times$ nearest-neighbor up-sampling, max pooling with 2×2 filters and stride 2, and batch normalization, respectively.

Below we provide additional examples of how to realize the pretraining loss $\mathcal{L}_{\text{RR}}$ in a neural network architecture. As illustrated in Equation 4.3, $\mathbf{d}_m$, and λ_m denote the vector of activated intermediate data in layer m from the forward pass, and a scaling factor, respectively. $\mathbf{d}'_m$ denotes the activations of layer m from the reverse pass. For instance, let $L_m()$ denote the operations of a layer m in the foward pass, and $L'_m()$ the corresponding operations for the reverse pass. Then $\mathbf{d}_{m+1} = L_m(\mathbf{d}_m)$, and $\mathbf{d}'_m = L'_m(\mathbf{d}'_{m+1})$.

When Equation 4.14 is minimized, we obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e. $\mathbf{d}'_{m+1} = \mathbf{d}_{m+1}$ holds. Then $\mathbf{d}'_m$ can be reconstructed from the incoming activations from the reverse pass, i.e., $\mathbf{d}'_{m+1}$, or from the output of layer m, i.e., $\mathbf{d}_{m+1}$. Using $\mathbf{d}'_{m+1}$ results in a global coupling of input and output throughout all layers, i.e., the *full* loss variant. On the other hand, $\mathbf{d}_{m+1}$ yields a variant that ensures local reversibility of each layer, and yields a very similar performance, as we will demonstrate below. We employ this *local* loss for networks without a unique, i.e., bijective, connection between two layers. Intuitively, when inputs cannot be reliably reconstructed from outputs.

Full Network Pretraining An illustration of a CNN structure with AFs and BN and a full loss is shown in Figure 4.2 in the main paper. To illustrate this setup, we consider an example network employing convolutions with mixed AFs, BN, and MP. Let the network receives a field of 32^2 scalar values as input. From this input, 20, 40, and 60 feature maps are extracted in the first three layers. Besides, the kernel sizes are decreased from 5×5 to 3×3. To clarify the structure, we use ReLU activation for the first convolution, while the second one uses a hyperbolic tangent, and the third one a sigmoid function. With the notation outlined above, the first three layers of the network are

$$\begin{aligned} I(32,32,1) = \mathbf{d}_1 &\rightarrow C_1(5,20,1) + \mathbf{b}_1 \rightarrow BN_1 \rightarrow \text{relu} \\ &\rightarrow \mathbf{d}_2 \rightarrow MP \rightarrow C_2(4,40,1) + \mathbf{b}_2 \rightarrow BN_2 \rightarrow \tanh \\ &\rightarrow \mathbf{d}_3 \rightarrow MP \rightarrow C_3(3,60,1) + \mathbf{b}_3 \rightarrow BN_3 \rightarrow \text{sigm} \\ &\rightarrow \mathbf{d}_4 \rightarrow ... \end{aligned} \tag{4.18}$$

The reverse pass for evaluating the loss re-uses all weights of the forward pass and ensures that all intermediate vectors of activations, $\mathbf{d}_m$ and $\mathbf{d}'_m$, have the same size and content in terms of normalization and non-linearity. We always consider states after activation for $\mathcal{L}_{\text{RR}}$. Thus, $\mathbf{d}_m$ denotes activations before pooling in the forward pass and $\mathbf{d}'_m$ contains data after up-sampling in the reverse pass, in order to ensure matching dimensionality. Thus, the last three layers of the reverse network for computing $\mathcal{L}_{\text{RR}}$ take the form:

$$\begin{aligned} ... &\rightarrow \mathbf{d}'_4 \rightarrow -\mathbf{b}_3 \rightarrow D_3(3,40,1) \rightarrow BN_2 \rightarrow \tanh \rightarrow UP \\ &\rightarrow \mathbf{d}'_3 \rightarrow -\mathbf{b}_2 \rightarrow D_2(4,20,1) \rightarrow BN_1 \rightarrow \text{relu} \rightarrow UP \\ &\rightarrow \mathbf{d}'_2 \rightarrow -\mathbf{b}_1 \rightarrow D_1(5,3,1) \\ &\rightarrow \mathbf{d}'_1 = O(32,32,1). \end{aligned} \tag{4.19}$$

Here, the de-convolutions D_x in the reverse network share weights with C_x in the forward network. I.e., the $4 \times 4 \times 20 \times 40$ weight matrix of C_2 is reused in its transposed form as a $4 \times 4 \times 40 \times 20$ matrix in D_2. Additionally, it becomes apparent that AFs and BN of layer 3 from the forward pass do not appear in the listing of the three last layers of the reverse pass. This is caused by the fact that both are required to establish the latent space of the fourth layer. Instead, $\mathbf{d}_3$ in our example represents the activations after

the second layer (with BN_2 and $tanh$), and hence the reverse pass for $\mathbf{d}_3'$ reuses both functions. This ensures that $\mathbf{d}_m$ and $\mathbf{d}_m'$ contain the same latent space content in terms of range and dimensionality, and can be compared in Equation 4.14.

For the reverse pass, we additionally found it beneficial to employ an AF for the very last layer if the output space has suitable content. For instance, for inputs in the form of RGB data we employ an additional activation with a ReLU function for the output to ensure the network generates only positive values.

Localized Pretraining In the example above, we use a full pretraining with $\mathbf{d}_{m+1}'$ to reconstruct the activations $\mathbf{d}_m'$. However, if the architecture of the original network makes use of operations between layers that are not bijective, e.g., residual connections, we instead use the local loss. Note that our loss formulation has no problems with irreversible operations within a layer, e.g., most convolutional or fully-connected layers typically are not fully invertible. In all these cases the loss will drive the network towards a state that is as-invertible-as-possible for the given input dataset. However, this requires a reliable vector of target activations in order to apply the constraints. If the *connection* between layers is not bijective, we cannot reconstruct this target for the constraints, as in the examples given above. In such cases, we regard every layer as an individual unit to which we apply the constraints by building a localized reverse pass. For example, given a simple convolutional architecture with

$$\mathbf{d}_1 \rightarrow C_1(5, 20, 1) + \mathbf{b}_1 = \mathbf{d}_2 \tag{4.20}$$

in the forward pass, we calculate $\mathbf{d}_1'$ with

$$(\mathbf{d}_2 - \mathbf{b}_1) \rightarrow D_1(5, 3, 1) = \mathbf{d}_1'. \tag{4.21}$$

We, e.g., use this local loss in the Resnet110 network below. It is important to note that despite being closer to regular autoencoder pretraining, this formulation still incorporates all non-linearities of the original network structure, and jointly trains full networks while taking into account the original learning objective.

4.4 Application

We now turn to a broad range of network structures, i.e., CNNs, Autoencoders, and GANs, with a variety of datasets and tasks to show our approach succeeds in improving inference accuracy and generality for modern-day applications and architectures. All tests use non-linear activations and several of them include BN. Experimental details are provided in Appendix B.

CIFAR-100 classification We first focus on orthogonalization for a CIFAR-100 classification task with a ResNet-18 network, and compare the performance of RR with the variants Std, Ort, in addition to an OCNN (in light blue) network [179]. The CNN architecture has ca. 11 million trainable parameters in each case. Pre is not included

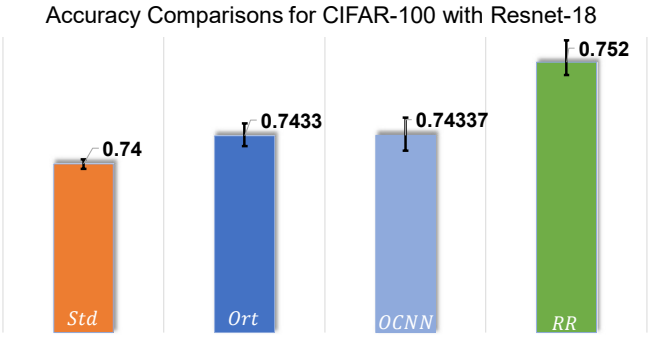


Figure 4.7: CIFAR-100 classification performance for RR, Std, Ort and OCNN. RR yields the highest accuracy, and outperforms state-of-the-art methods for orthogonalization (Ort and OCNN).

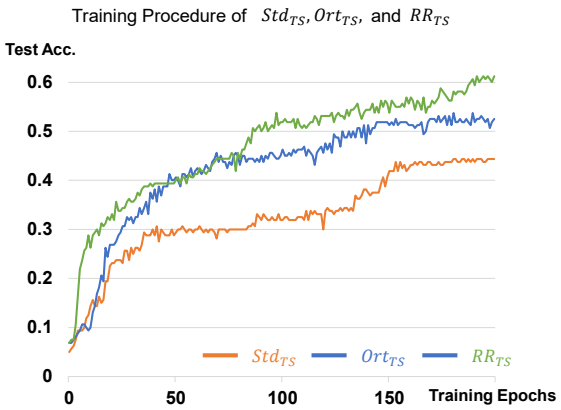


Figure 4.8: Test accuracy over training epochs for Std_{TS}, Ort_{TS}, and RR_{TS}. The RR_{TS} model consistently exhibits faster convergence than the other two versions.

in this comparison due to its incompatibility with ResNet architectures. The resulting performance for the different variants (evaluated for 3 runs each) is shown in Figure 4.7. For CIFAR-100, the orthogonal regularizations (Ort and OCNN) result in noticeable performance gains of 0.33% and 0.337%, but RR clearly outperforms both with an improvements of 1.2%. Despite being different formulations, both Ort and OCNN represent orthogonal regularizers that aim for the same goal of weight orthogonality. Hence, their performance is on-par, and we will focus on the more generic Ort variant for the following evaluations.

Transfer-learning Benchmarks We evaluate our approach with two state-of-the-art benchmarks for transfer learning. The first one uses the *texture-shape* dataset from [180], which contains challenging images of various shapes combined with patterns and textures to be classified. The results below are given for 10 runs each. For the stylized data shown in Figure 4.9 (a), the accuracy of Pre_{TS} is low with 20.8%. This result is in line with observations in previous work and confirms the detrimental effect of classical

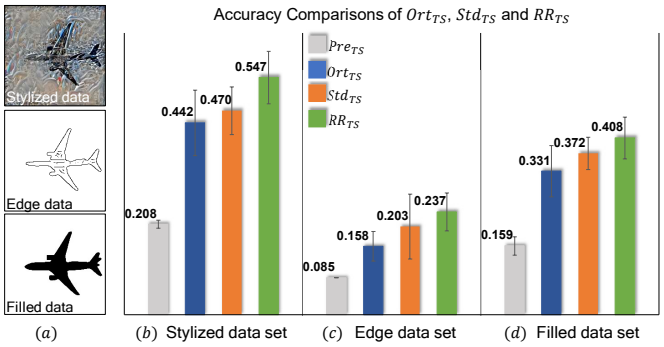


Figure 4.9: a) Examples from texture-shape dataset. **b, c, d)** Texture-shape test accuracy comparisons of Pre_{TS}, Ort_{TS}, Std_{TS} and RR_{TS} for different datasets.

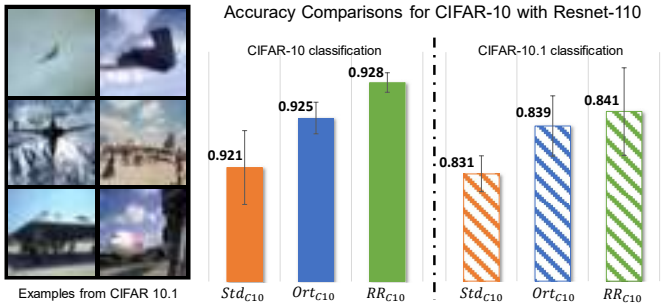


Figure 4.10: Left: Examples from CIFAR 10.1 dataset. Right: Accuracy comparisons when applying models trained on CIFAR 10 to CIFAR 10.1 data.

pretraining. Std_{TS} yields a performance of 44.2%, and Ort_{TS} improves the performance to 47.0%, while RR_{TS} yields a performance of 54.7% (see Figure 4.9 b). Thus, the accuracy of RR_{TS} is 162.98% higher than Pre_{TS}, 23.76% higher than Std_{TS}, and 16.38% higher than Ort_{TS}. To assess generality, we also apply the models to new data without re-training, i.e. an edge and a filled dataset, also shown in Figure 4.9 (a). For the edge dataset, RR_{TS} outperforms Pre_{TS}, Std_{TS} and Ort_{TS} by 178.82%, 50% and 16.75%, respectively.

Exemplary curves for test accuracy at training time for Std_{TS}, Ort_{TS}, and RR_{TS} are shown in Figure 4.8. Pre_{TS} is not included since it's layer wise curriculum precludes a direct comparison. The graph shows that RR_{TS} converges faster than Std_{TS} and Ort_{TS} from the very beginning. It achieves the performance of Std_{TS} and Ort_{TS} with ca. $\frac{1}{3}$ and $\frac{1}{2}$ of number of training epochs, respectively. Achieving comparable performance with less training effort, and a higher final performance support the reasoning given in Sec. 4.1: RR_{TS} with its reverse pass is more efficient at extracting relevant features from the training data. Over the course of our tests, we observed a similar convergence behavior for a wide range of other runs.

It is worth pointing out that the additional constraints of our training approach lead to moderately increased requirements for memory and computations, e.g., 41.86% more

time per epoch than regular training for the texture-shape test. On the other hand, it allows us to train smaller models: we can reduce the weight count by 32% for the texture-shape case while still being on-par with Ort_{TS} in terms of classification performance. By comparison, regular layer-wise pretraining requires significant overhead and fundamental changes to the training process. Our pretraining fully integrates with existing training methodologies and can easily be deactivated via $\lambda_m = 0$. More details of runtime performance and training behavior are given in the appendix.

As a second test case, we use a CIFAR-based task transfer [181] that measures how well models trained on the original CIFAR 10, generalize to a new dataset (CIFAR 10.1) collected according to the same principles as the original one. Here we use a Resnet-110 with 110 layers and 1.7 million parameters, Due to the consistently low performance of the Pre models [182], we focus on Std, Ort and RR for this test case. In terms of accuracy across 5 runs, Ort_{C10} outperforms Std_{C10} by 0.39%, while RR_{C10} outperforms Ort_{C10} by another 0.28% in terms of absolute test accuracy (Figure 4.10). This increase for RR training matches the gains reported for orthogonality in previous work [166], thus showing that our approach yields substantial practical improvements over the latter. It is especially interesting how well performance for CIFAR 10 translates into transfer performance for CIFAR 10.1. Here, RR_{C10} still outperforms Ort_{C10} and Std_{C10} by 0.22% and 0.95%, respectively. Hence, the models from our pretraining very successfully translate gains in performance from the original task to the new one, indicating that the models have successfully learned a set of more general features. To summarize, both benchmark cases confirm that the proposed pretraining benefits generalization.

4.4.1 Smoke Generation

In this section, we employ our pretraining in the context of generative models for transferring from synthetic to real-world data from the ScalarFlow dataset [183]. As SR task A, we first use a fully-convolutional generator network, adversarially trained with a discriminator network on the synthetic flow data. While regular pretraining is more amenable to generative tasks than orthogonal regularization, it can not be directly combined with adversarial training. Hence, we pretrain a model Pre for a reconstruction task at HR without a discriminator instead. Figure 4.11(a) demonstrates that our method works well in conjunction with the GAN training: As shown in the bottom row, the trained generator succeeds in recovering the input via the reverse pass without modifications. A regular model Std_{A}, only yields a black image in this case. For Pre_{A}, the layer-wise nature of the pretraining severely limits its capabilities to learn the correct data distribution [184], leading to low performance.

We now mirror the generator model from the previous task to evaluate an autoencoder structure that we apply to two different datasets: the synthetic smoke data used for the GAN training (task B_1), and a real-world RGB dataset of smoke clouds (task B_2). Thus both variants represent transfer tasks, the second one being more difficult due to the changed data distribution. The resulting losses, summarized in Figure 4.11(b), show that RR training performs best for both autoencoder tasks: the L^2 loss of RR_{AB_1} is 68.88% lower than Std_{AB_1}, while it is 13.3% lower for task B_2. The proposed pretraining also

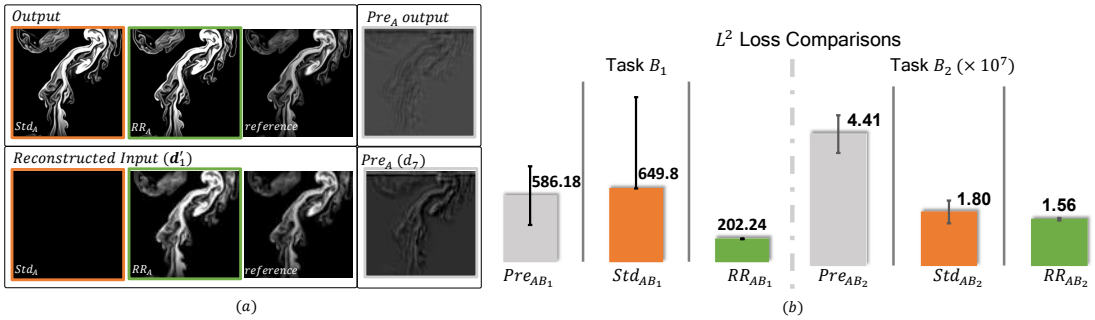


Figure 4.11: a) Example output and reconstructed inputs, with the reference shown right. Only RR_A successfully recovers the input, Std_A produces a black image, while Pre_A fares poorly. **b)** L^2 loss comparisons for two different generative transfer learning tasks (averaged across 5 runs each). The RR models show the best performance for both tasks.

clearly outperforms the Pre variants. Within this series of tests, the RR performance for task B_2 is especially encouraging, as this task represents a synthetic to real transfer.

4.4.2 Weather Prediction

Pretraining is particularly attractive in situations where the amount of data for training is severely limited. Weather forecasting is such a case, as accurate, real-world data for many relevant quantities are only available for approximately 50 years. We use the ERA dataset [145] consisting of assimilated measurements, and additionally evaluate our models with simulated data from the CMIP database [185]. We replicate the architecture and training procedure of the WeatherBench benchmark [186]. Hence we use prognostic variables at seven vertical levels, together with some surface and constant fields at the current time t as well as $t - 6h$ and $t - 12h$ as input, and target three-day forecasts of 500 hPa geopotential (Z500), 2-meter temperature (T2M), and 850 hPa temperature (T850). We use a convolutional ResNet architecture with 19 residual blocks and 6.36M trainable parameters, with a latitude-weighted root mean squared error (RMSE) as loss functions for training. For the worldwide observations dataset ERA (six-hour intervals with a 5.625° resolution.), we train the models with data from 1979 to 2015 and evaluate performance with RMSE measurements across all data points from the years 2017 and 2018. For the historical simulation dataset CMIP, the years 1850 to 2005 are used as training data, while performance is measured with the years 2006 to 2014.

We show comparisons between the regular model Std and RR for both ERA and CMIP datasets. As Rasp et al. [186] relied on dropout regularization, we additionally train and evaluate models for both datasets with and without dropout. Following their methodology, L_2 regularization is applied for all tests. As regular pretraining does not support residual connections, we omit it for the weather forecasting tests.

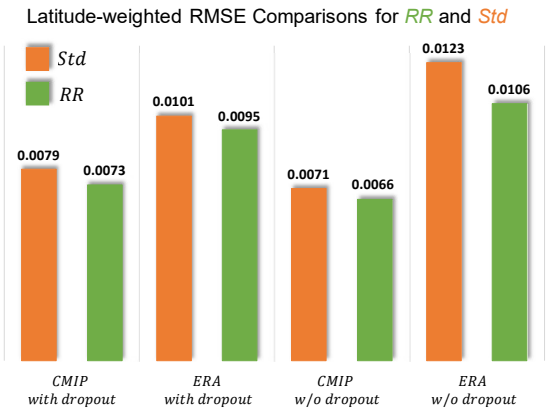


Figure 4.12: Latitude-weighted RMSE comparisons between Std and RR for ERA and CMIP datasets. Models trained with RR pretraining significantly outperform state-of-the-art Std for all cases. The minimum performance improvements of RR is 5.7% for the case with ERA dataset and dropout regularization.

Performance comparisons are shown in Figure 4.12. Across all cases, irrespective of whether observation data or simulation data is used, the RR models clearly outperform the regular models and yield consistent improvements. This also indicates that our approach is compatible with other forms of regularization, such as dropout and L_2 regularization. The RR models yield performance improvements of 6% ˜8% for the CMIP cases, and the ERA case with dropout. Here the re-trained Std version is on-par with the data reported in [186], while our RR model exhibits a performance improvement of 6.3% on average. For the ERA dataset without dropout regularization, the RR model decreases the loss even more strongly by 13.7%.

Visualizations of an inference result for 9 Aug. 2019 22:00 for the ERA dataset without dropout regularization are shown in Figure 5.1 and Figure 4.13(a). Predictions of RR yield lower errors, and are closer to the reference. The same conclusions can be drawn from the example at 26 June 2014 0:00 from the CMIP dataset without dropout regularization in Figure 4.13(b).

4.5 Discussion and Conclusions

We have proposed a novel pretraining approach inspired by classic methods for unsupervised autoencoder pretraining and orthogonality constraints. In contrast to the classical methods, we employ a constrained reverse pass for the full non-linear network structure and include the original learning objective. Weight matrix SVD is applied to visually analyse and interpret that our proposed method is more capable of extracting dominant features from the training dataset. We have shown for a wide range of scenarios, from mutual information, over transfer learning benchmarks to weather forecasting, that the proposed pretraining yields networks with improved performance and better generalizing capabilities. Our training approach is general, easy to integrate, and imposes no

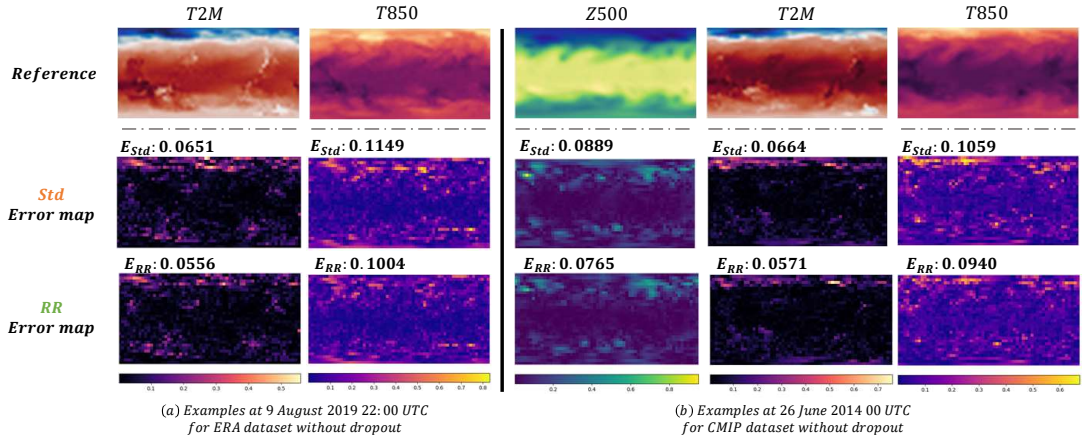


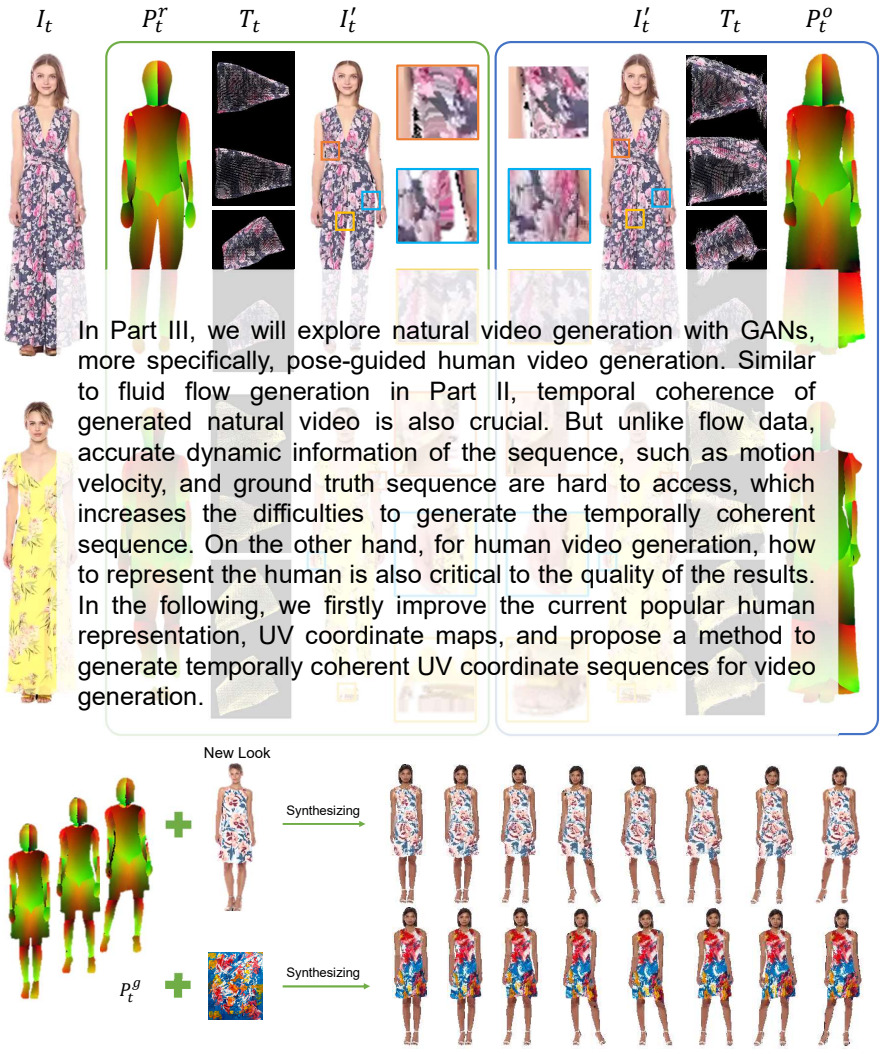
Figure 4.13: a) Comparisons of predictions for T2M and T850 on 9 Aug. 2019, 22:00 for the ERA dataset without dropout regularization. **b)** Prediction comparisons of three physical quantities on 26 June 2014, 0:00 for the CMIP dataset without dropout regularization. As confirmed by the quantified results, RR predicts results closer to the reference.

requirements regarding network structure or training methods. As a whole, our results show that unsupervised pretraining has not lost its relevance in today's deep learning environment.

As future work, we believe it will be exciting to evaluate our approach in additional contexts, e.g., for temporal predictions [187, 188], and for training explainable and interpretable models [189, 177, 190].

Part III

Natural Video Generation with GANs



In Part III, we will explore natural video generation with GANs, more specifically, pose-guided human video generation. Similar to fluid flow generation in Part II, temporal coherence of generated natural video is also crucial. But unlike flow data, accurate dynamic information of the sequence, such as motion velocity, and ground truth sequence are hard to access, which increases the difficulties to generate the temporally coherent sequence. On the other hand, for human video generation, how to represent the human is also critical to the quality of the results. In the following, we firstly improve the current popular human representation, UV coordinate maps, and propose a method to generate temporally coherent UV coordinate sequences for video generation.

5 Temporally Coherent UV Coordinate Generation

In image or video generation tasks [105, 106] that involve people, it is crucial to obtain accurate representations of the 3D human shape and appearance to efficiently generate modified content. In this context, *UV coordinates* are a popular 2D representation that establish dense correspondences between 2D images and 3D surface-based representations of the human body. UV coordinates go beyond skeleton landmarks to encode human pose and shape, and are widely used in image/video editing, augmented reality, and human-computer interaction [191, 192, 193]. In this chapter, we tackle video generation of people, with a focus on efficiency and capturing loose clothing. Unlike previous works [132, 96, 194] which use large networks to capture motion and appearance, we train a model to generate temporally coherent UV coordinates. We use a single, fixed texture to store appearance information so that our model can solely focus on learning UV dynamics.

Human body UV coordinates can be derived indirectly from estimates of 3D shape models [110, 111, 112, 113] like SMPL [114]. Alternatively, direct estimation methods like DensePose [16] and UltraPose [115] bypass intermediate 3D models to directly output UV coordinates from a single RGB image. The convenience of direct methods has led to DensePose being widely used in animation and editing applications [109, 116, 117, 118]. Nevertheless, the UV coordinates obtained from SMPL and DensePose approximate only human body silhouettes in tight clothing. They do not capture loose clothing, such as long skirts or wide pants (see comparisons in Figure 5.6 and 5.7). In addition, the methods for UV estimation work only on individual images. For video inputs, they are applied frame-by-frame [119, 120] without considering the temporal relationship between frames. As such, the UV coordinates are inconsistent over time, so any re-targeted sequences will shift and jitter.

In this chapter, we focus on improving the spatial coverage and temporal coherence of UV coordinates generated from a sequence of 2D images. We target the ability to retain the full body plus clothing silhouette for arbitrary styles of clothing. Our approach is agnostic to the UV source, which we demonstrate via inputs from both DensePose [16] and SMPL model estimates [113]. For temporal coherence, we aim at achieving the

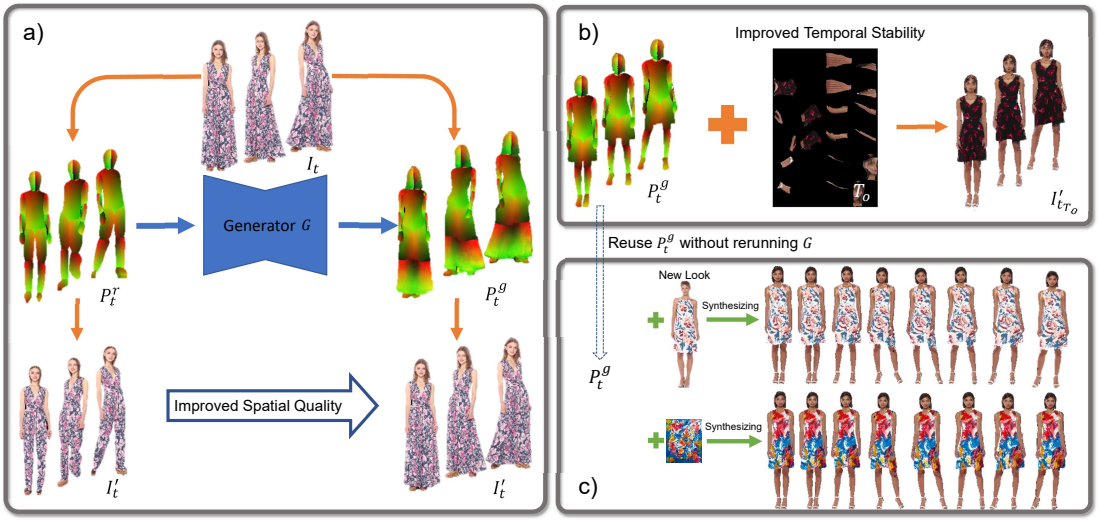


Figure 5.1: a) Our method generates temporally coherent UV coordinates that capture loose clothing from off-the-shelf human pose UV estimates such as SMPL and DensePose [114, 16]. b) Generated UV coordinates allow us to recover entire sequences from a constant texture map. c) Virtual try-on and modifications of the look can be easily achieved with minimal computation via a simple lookup.

point-to-point correspondences among different frames via UV coordinate maps, so that video sequences can be generated with one fixed texture.

A core challenge of learning a model for extended and temporally coherent UV coordinates lies in the lack of data for direct supervision. Hence, we propose a novel learning scheme that combines both supervised and unsupervised components. We first pre-process a sequence of UV coordinates obtained from DensePose or SMPL via spatial extension and temporal stabilization to obtain training data for an initial training stage. We then shift the learning gradually from supervised, with the pre-processed data, to unsupervised, driven by a differentiable UV mapping pipeline between the texture and image space.

Our results demonstrate that using loss terms formulated in both UV and image space are crucial for generating high-quality UV coordinates with temporal coherence. As our generator does not take RGB images as input, the UV coordinates generated from our trained model can be directly paired with different texture maps to generate virtual try-on videos with a very simple lookup step. This is device-independent and orders of magnitude more efficient than other methods, which generate video outputs by evaluating neural networks.

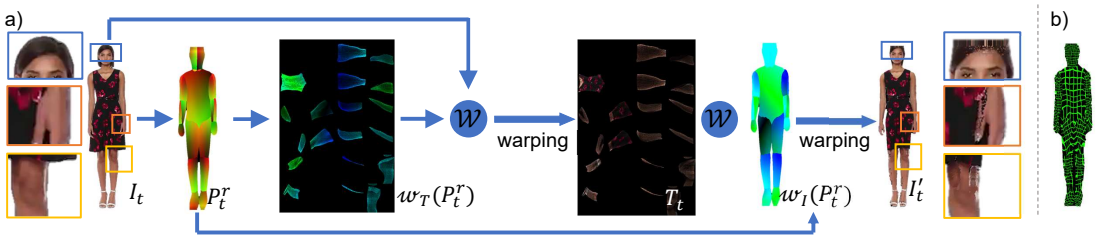


Figure 5.2: a) Example mapping from I_t to T_t via P_t^r, and back to I_t'. P_t^r cannot fully recover the image, and misses skirt, hair, and shoulder parts. Besides, colours inside the human body are also partially incorrect. b) Mapping results of P_t^r with T_{grid} as input. Most quadrants are preserved, indicating that the corresponding features are not destroyed after the UV mapping.

5.1 Preliminaries

5.1.1 Notation & definitions

An image $I_t \in \mathbb{R}^{s_x \times s_y \times 3}$ for frame t in a sequence stores RGB information at a location $\mathbf{x} \in \mathbb{R}^{s_x \times s_y}$. The appearance of a person in I_t can also be represented in a texture $T_t \in \mathbb{R}^{t_x \times t_y \times 3}$ with locations $\mathbf{u}$. The image I_t and texture T_t are related via the UV coordinates $P_t \in \mathbb{R}^{s_x \times s_y}$, where

$$P_t(\mathbf{x}) = \mathbf{u}, \quad \text{s.t.} \quad I_t(\mathbf{x}) = T_t(\mathbf{u}). \tag{5.1}$$

To ensure differentiability, we treat I_t, P_t and T_t as continuous functions in space via a suitable interpolation operator; we use bi-linear interpolation in our work. In practice, the three fields are represented as time sequences over t.

The corresponding texture T_t for an image I_t can be generated by warping I_t with function $\mathcal{W}$ via the warping grid $\omega_T(P_t)$; conversely, the image content can also be recovered as I_t' from the texture T_t and UV coordinates P_t with warping grid ω_I from T_t to I_t (see Figure 5.2):

$$T_t = \mathcal{W}(I_t,\, \omega_T(P_t)) \quad \text{and} \quad I_t' = \mathcal{W}(T_t,\, \omega_I(P_t)). \tag{5.2}$$

Note the warping function $\mathcal{W}(I, \omega)$, for every location $\mathbf{x}$ in I, returns a bi-linear interpolation of I at location $\omega(\mathbf{x})$.

In our work, we refer to the UV outputs from DensePose [16] or unwrapped from the 3D mesh of models like SMPL [113] as *raw* UV coordinates, denoted by P_t^r for frame t. Raw UV coordinates are typically restricted by the human body silhouette. As such, loose clothing parts are cut off (see the missing skirt parts in Figure 5.1a and Figure 5.2a. Additionally, the raw UV P_t^r is not one-to-one. Multiple pixels $\mathbf{x}$ of I_t may be mapped to the same $\mathbf{u}$ in T_t, leading to a loss of information in T_t. These two shortcomings may result in extreme and undesirable differences between the original I_t and the reconstructed I_t' (see example in Figure 5.2a. For a sequence of images over time, the differences are further compounded. As P_t^r can only be estimated frame-wise, resulting textures T_t tend to lack correspondence over time.

5.1.2 Problem formulation

Given the non-idealities of P_t^r, we aim to develop a system that can output a sequence of refined UV coordinates P_t^g leading to faithful reconstructions $I_t' = I_t$. Additionally, we aim for an independent and lightweight appearance representation in the form of a single texture T_o, which is constant over time.

We start by defining a model G parameterized by θ to estimate refined UV coordinates P_t^g from raw UV P_t^r:

$$P_t^g = G(P_t^r; \theta). \tag{5.3}$$

For I_t' to be of high quality and for P_t^g to be temporally stable, we consider appearance and temporal loss functions

$$\begin{aligned} \mathcal{L}_{\text{app}} &= \sum_{t=0}^{N} (||I_t' - I_t||^2) = \sum_{t=0}^{N} (||\mathcal{W}(T_t, \omega_I(P_t^g)) - I_t||^2), \\ \mathcal{L}_{\text{temp}} &= \sum_{t=0}^{N} (||T_t - T_o||^2) = \sum_{t=0}^{N} (||\mathcal{W}(I_t, \omega_T(P_t^g)) - T_o||^2), \end{aligned} \tag{5.4}$$

where N represents the sequence length and T_o a constant texture. Minimizing $||I_t' - I_t||^2$ leads to improvements of I_t'. Minimizing $||T_t - T_o||^2$ encourages a constant texture, which in turn largely alleviates inconsistent correspondences over time.

5.2 Method

One could learn θ of model G if raw UV (P_t^r) were paired ground truth UV coordinates fulfilling the constraints in Equation 5.4. Such ground truth data does not exist in practice, so we are forced to consider indirect approaches. Naively applying an unsupervised or self-supervised training is ill-conditioned and error-prone, due to the strong non-linearities in mappings between I_t, T_t, and P_t. As such, we propose an approach to combine both supervised and unsupervised learning.

We start with a data pre-processing step (Sections 5.2.1 to 5.2.3) that gradually refines P_t^r to establish "ground-truth". It is worth noting that we handle the two parts of Equation 5.4 separately due to the strong non-linearity and large distance between P_t^r and P_t^g. After an initial training of G with the pre-processed data, we then incorporate unsupervised losses from the image space (Section C.1) to train a final model G that jointly improves spatial and temporal quality. The trained model G generates full-silhouette UV coordinates for different poses.

Since appearance or RGB information is encoded only in the texture T_o, which is used for the loss and preprocessing of the data but not a part of the network inputs, the resulting UV coordinate sequence P_t^g can be directly used for video generation with any given texture. Subsequently, generating a new output sequence with changed colours or patterns is highly efficient.

5.2.1 UV extension

Raw UV inputs omit important details (see example in Figure 5.2a. First, we aim to achieve full silhouette coverage for P_t^r. To better understand the relationship between I_t, P_t^r and T_t, we visualize UV mapping results for a synthetic grid texture $\mathcal{W}(T_{grid}, \omega_I(P_t^r))$ in Figure 5.2b. Here, T_{grid} contains an evenly distributed grid quadrants, which remain well-preserved, suggesting that the UV mapping with P_t^r retains a piece-wise regular surface manifold, albeit with different scaling factors.

The grid structure suggests that neighbouring points in I_t remain neighbours in T_t, and additional entries can be added to the raw UV coordinates P_t^r via extrapolation from neighbouring points. It is worth pointing out that for traditional UV generation, cutting the object surface and minimizing surface distortion are two challenging steps [195]. The raw UV coordinates provide an initial unwrapping of the body, hence we focus on solving the latter challenge of minimizing distortions when extrapolating content in the UV coordinates.

In this chapter, we extend the UV coordinates through energy minimization, employing a *virtual mass-spring* system. Mass-spring systems are commonly used in the simulation of clothing [196, 50]. Additionally, [197] and [198] have shown that the potential energy of a mass-spring system is minimized at the equilibrium state. We map the new extrapolated point from I_t to T_t and apply a *virtual mass-spring* system in T_t to reduce the surface distortion. We assume that all neighbouring points $O_1, O_2, ..., O_n$ inside a region of size 40×40 are connected with this new extrapolated point O_0 via *virtual* springs in the texture. The pushing/pulling forces $\mathbf{f_1}, \mathbf{f_2}, ..., \mathbf{f_n}$ from neighbour points will drive O_0 to the direction of $\sum_{i=1}^{n} \mathbf{f_i}$ for every step until O_0 arrives at an equilibrium state with a new position, where $\sum_{i=1}^{n} \mathbf{f_n} = \mathbf{0}$. From our experience, we found that applying pure pushing forces can generate valid results, since the parts that need to be extended are always located at the outline of the body. After applying pushing forces, we switch the forces to pulling in order to make the texture more compact. In our formulation, springs naturally encode the area preservation constraints among new extrapolated points and their neighbouring points in a small region of the texture map. The spring forces drive the new extrapolated points to new positions until the system finds an equilibrium state with reduced distortion.

We denote the UV coordinates after the extension with P_t^e. An example result is shown in Figure 5.3b. We can see that the missing parts from the raw UV coordinates computed via DensePose are recovered successfully, such as the side of the dress.

5.2.2 UV optimization

After UV extension, artifacts in I_t' may remain (See Figure 5.3b). One cause of these artifacts is duplicate UV coordinates in P_t^r, as it is not constrained to be a one-to-one mapping, especially for direct methods such as DensePose. To further improve P_t, we directly minimize $\mathcal{L}_{\text{app}}(P_t)$ via gradient descent, initializing P_t with the extended UV map P_t^e. The gradient $\frac{\partial \mathcal{L}_{\text{app}}(P_t)}{\partial P_t}$ can be estimated via the intermediate warping grids

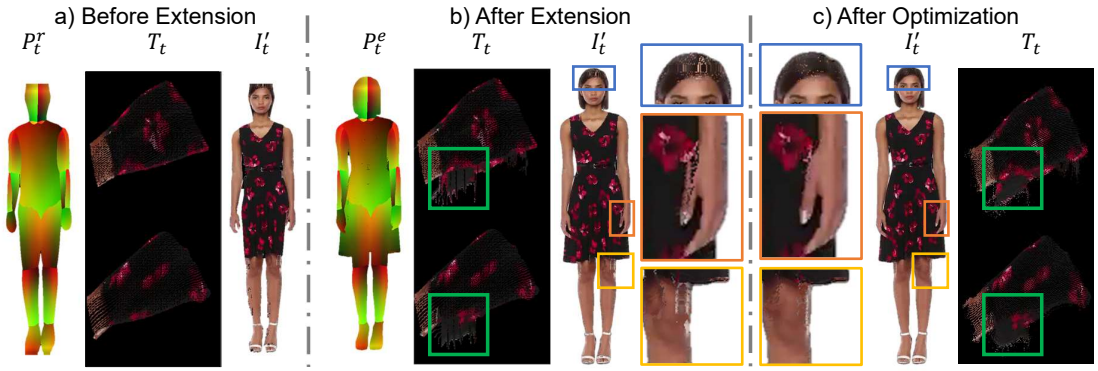


Figure 5.3: a) Raw UV coordinates, b) with application of UV extension and c) optimization. The UV extension allows missing parts such as the dress to be mapped into the correct parts of T_t, while UV optimization makes I'_t closer to I_t.

$\omega_T(P_t^r)$, and $\omega_I(P_t^r)$. More Specifically, from $\mathcal{L}_{\text{app}}(P_t)$ we will have

$$\frac{\partial \mathcal{L}_{\text{app}}}{\partial P_t} = \frac{\partial \mathcal{L}_{\text{app}}}{\partial I'_t} \times \frac{\partial I'_t}{\partial P_t}. \tag{5.5}$$

We compute the gradient $\frac{\partial I'_t}{\partial P_t}$ via the intermediate warping grids $\omega_T(P_t^r)$ and $\omega_I(P_t^r)$ from UV mappings

$$T_t = \mathcal{W}(I_t,\, \omega_T(P_t)) \quad \text{and} \quad I'_t = \mathcal{W}(T_t,\, \omega_I(P_t)). \tag{5.6}$$

And relationship between $\omega_I(P_t)$ and P_t can be written as

$$\omega_I(P_t(\mathbf{x})) = \mathbf{x} - P_t(\mathbf{x}). \tag{5.7}$$

This gives:

$$\begin{aligned} \frac{\partial I'_t}{\partial P_t} &= \frac{\partial I'_t}{\partial T_t} \times \frac{\partial T_t}{\partial P_t} + \frac{\partial I'_t}{\partial \omega_I(P_t)} \times \frac{\partial \omega_I(P_t)}{\partial P_t}; \\ \frac{\partial T_t}{\partial P_t} &= \frac{\partial T}{\partial \omega_T(P_t)} \times \frac{\partial \omega_T(d(I_t))}{\partial P_t}; \\ \frac{\partial \omega_T(P_t)}{\partial P_t} &= \frac{\omega_T(P_t)}{\partial \omega_I(P_t)} \times \frac{\partial \omega_I(P_t)}{\partial P_t}. \end{aligned} \tag{5.8}$$

In an implementation, we can conveniently obtain $\frac{\partial \omega_T(P_t)}{\partial P_t}$ via

$$\frac{\partial \omega_T(P_t)}{\partial P_t} = \mathcal{W}(\frac{\partial \omega_I(P_t)}{\partial P_t}, \omega_T(P_t)), \tag{5.9}$$

and $\frac{\partial \omega_I(P_t)}{\partial P_t}$ can be computed via Equation 5.7. This provides $\frac{\partial I'_t}{\partial P_t}$ for optimization and learning steps.

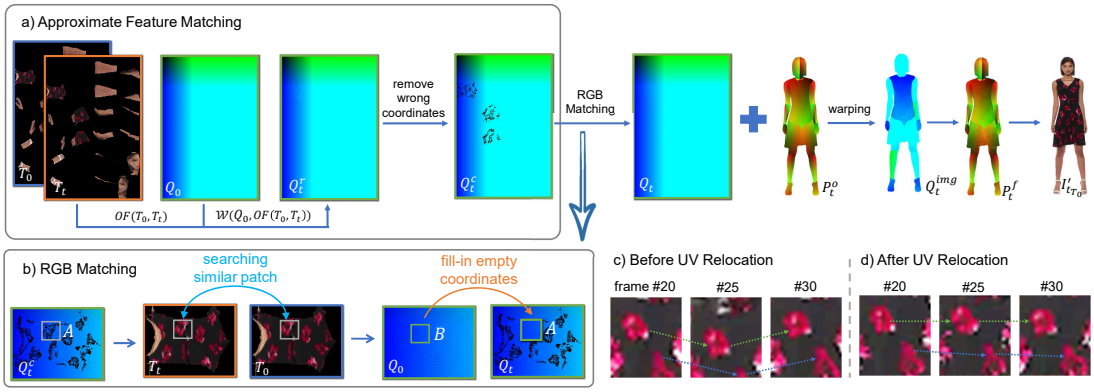


Figure 5.4: Overview and results of temporal UV generation. a) Approximate feature matching is achieved via the optical flow (OF) from T_o to T_t. b) RGB matching is applied to correct the coordinates resulting from errors in OF. Images in c) are generated with P_t^o and T_o, i.e., $I'_{t_{T_o}} = \mathcal{W}(T_o, \omega_I(P_t^o))$. Images in d) are similarly generated with P_t^f. Green and blue arrows are shown here to track the two patterns in the images. After the temporal relocation step, results are more temporally coherent.

Following common practice in non-linear settings, we add a gradient and Laplacian regularizer to encourage smooth solutions [199] and minimize $\mathcal{L}_{\text{app}} + L_r$, where

$$L_r = \alpha_1(||\nabla P_t||_F^2) + \alpha_2 \sum_{i,j=0,1} ||H_{ij}(P_t)||_F^2, \tag{5.10}$$

H is the Hessian and $||\cdot||_F$ denotes the Frobenius norm. It is visible in Figure 5.3c that most of the artifacts in I'_t have been removed by the optimization procedure, and the image content is significantly closer to the reference. We denote the optimized UVs with P_t^o.

We apply a gradient descent optimizer with $\alpha_1 = 100$ and $\alpha_2 = 10$ for regularizer L_r. Due to the large distance between P_t^e and P_t^o, we use a large learning rate, such as 10.0, to accelerate the optimization procedure. We found that promising results can be obtained after ca. 16500 steps. UV optimization takes about 75s/frame, measured for resolution 1200×800 with a NVIDIA RTX 2080 Ti GPU. All frames can be optimized in parallel.

5.2.3 UV temporal relocation

Minimizing $\mathcal{L}_{\text{temp}}$ in Equation 5.4 will improve the temporal stability of the texture maps. To do so, we find point correspondences $Q_t(\mathbf{u})$ between T_t and T_o so that $T_t(\mathbf{u}) = T_o(Q_t(\mathbf{u}))$. The correspondences allow new UV coordinates P_t^f to map I_t back to the constant T_o instead of T_t. For simplicity, we assign as the constant T_o the texture from frame 0 of a sequence, i.e., $T_o = T_0$.

We initialize the point correspondences with optical flow from T_o to T_t, i.e., $OF(T_o, T_t)$, as shown in Figure 5.4a. An approximate correspondence between T_t and T_o can be written as $Q_t^r(\mathbf{u}) = \mathcal{W}(Q_0(\mathbf{u}), OF(T_o, T_t))$. In theory, the reconstruction T_t' can then be reconstructed from T_o and $Q_t^r(\mathbf{u})$ via a *lookup* step, i.e., $T_t'(\mathbf{u}) = T_o(Q_t^r(\mathbf{u}))$.

Note that errors in $OF(T_o, T_t)$ makes $Q_t^r(\mathbf{u})$ only an approximate correspondence, and there are still differences between the reconstructed T_t' and the true T_t. To correct these errors, we remove the coordinates in $Q_t^r(\mathbf{u})$ where the texture content does not match, i.e., $T_t'(\mathbf{u}) \neq T_t(\mathbf{u})$. We then fill them in with regions from T_o to obtain the final $Q_t(\mathbf{u})$. The filling is based on a simple similarity measure of the RGB values. Specifically, we assume that similar, nearby texture patches in T_0 and T_t will have the same correspondences in Q_0 and Q_t. For a missing area A in Q_t^c , as shown in Figure 5.4, we locate the region with the same position as A in T_t. We record the values of Q_t^c and T_t inside region A with $[Q_t^c]_A$ and $[T_t]_A$, respectively. Then we can find a region B in T_0 via

$$min\ ||[T_t]_A - [T_0]_B||_F^2, \tag{5.11}$$

and $[Q_0]_B$ are used to fill in $[Q_t^c]_A$ to obtain Q_t.

Afterwards, Q_t can be mapped to Q_t^{img} in the image space via P_t^o. $Q_t^{img}(\mathbf{u})$ is directly our P_t^f if P_t^r is represented with the same coordinates system as $\mathbf{u}$, such as the P_t^r unwrapped from the SMPL model. But for the P_t^r from the DensePose model, which uses a different coordinate system from $\mathbf{u}$, we additionally transform $Q_t^{img}(\mathbf{u})$ into P_t^f.

Comparisons of results before and after the temporal relocation step are shown in Figure 5.4c-d. The images $I'_{t_{T_o}}$ recovered from T_o are more temporally coherent after the relocation step.

5.2.4 Temporal UV model training

So far, we have improved the spatial and temporal quality of the raw UV P_t^r separately. We now consider the two objectives jointly in a spatio-temporal manner and apply an adversarial training for G from Equation 5.3. The learned G can then generate complete UV coordinates P_t^g at test time given raw UV coordinates P_t^r. Below, we define several unsupervised loss terms in both the UV and RGB image space to guide the training and produce high-quality outputs.

Spatial loss (UV space). Recall that P_t^f is now the UV coordinates that relate image I_t to the constant texture T_o based on the UV relocation step in Sec. 5.2.3. We make use of a supervised L_2 loss

$$L_2 = \left\| G(P_t^r) - P_t^f \right\|_F^2 \tag{5.12}$$

and adversarial loss via a discriminator D_s:

$$\begin{aligned} L_s^{uv} &= -log(D_s(G(P_t^r))), \\ L_{D_s} &= -logD_s(P_t^f) - log(1 - D_s(G(P_t^r))). \end{aligned} \tag{5.13}$$

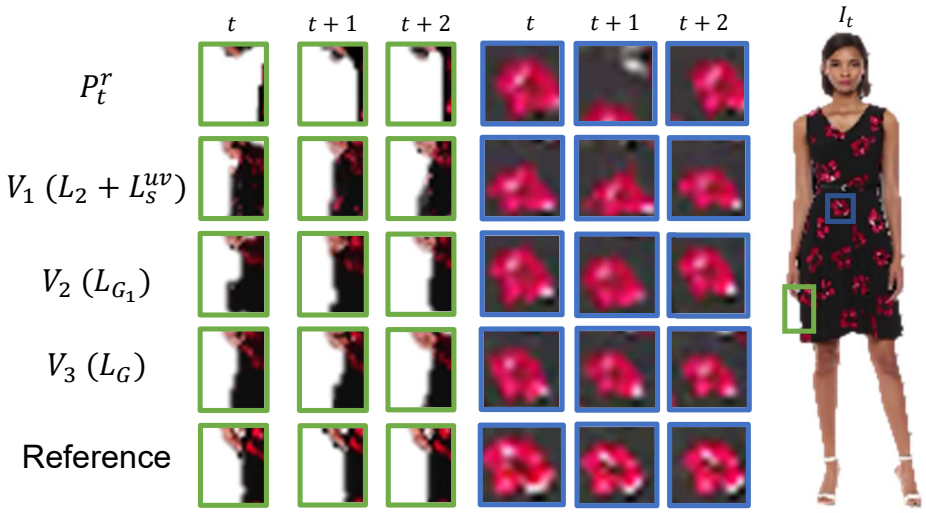


Figure 5.5: Comparisons of three successive frames, $(I'_{t-1_{T_o}}, I'_{t_{T_o}}, I'_{t+1_{T_o}})$, among results of P_t^r, V_1, V_2, and V_3. We show examples of the same region in the image to illustrate the temporal coherence of the generated videos. We can see that V_1 is more coherent than P_t^r because of the temporal relocation when preparing the training data. V_2 and V_3 show further improvements due to the temporal stability losses in UV and image spaces.

Temporal stability loss (UV space). We consider a smoothing loss between neighbouring frames $t-1$ and $t+1$:

$$\begin{aligned} L_{smo} = &\left\|G(P_{t-1}^r) - G(P_t^r)\right\|_F^2 + \left\|G(P_t^r) - G(P_{t+1}^r)\right\|_F^2 \\ &+ \left\|G(P_{t-1}^r) - 2 \times G(P_t^r) + G(P_{t+1}^r)\right\|_F^2, \end{aligned} \tag{5.14}$$

and add an unsupervised adversarial loss via a second discriminator network D_t:

$$\begin{aligned} L_t^{uv} = &- log(D_t(G(P_{t-1}^r), G(P_t^r), G(P_{t+1}^r))), \\ L_{D_t} = &- log(D_t(P_{t-1}^f), f(P_{t-1}^f), f(f(P_{t-1}^f))) \\ &- log(1 - D_t(G(P_{t-1}^r), G(P_t^r), G(P_{t+1}^r)), \end{aligned} \tag{5.15}$$

where f are randomized geometric transformations (e.g., translation, rotation or scaling). Note that ground truth over time is not available in our setting. We synthesize ground truth by randomly choosing a transformation f and applying it to P_{t-1}^f. This yields a reference for time t; applying the transformation again at $t+1$ yields an additional reference to form a synthetic triplet. The triplets serve as ground truth for the adversarial training of Equation 5.15 and guide the generation of smooth UV coordinates over time.

Spatial loss (image space). With the mapping pipeline from UV coordinates to images, an image-based L2 loss is applied at training time:

$$L_s^{\text{img}} = \|I_{g_t} - I_t\|_F^2, \text{ where } I_{g_t} = \mathcal{W}(T_o, \omega_I(G(P_t^r))). \tag{5.16}$$

Temporal stability loss (image space). Similar to the UV space, we define a temporal adversarial loss via an additional discriminator D_{img} in image space:

$$\begin{aligned} L_t^{img} =& - logD_{img}(I_{g_{t-1}}, I_{g_t}, I_{g_{t+1}}), \\ L_{D_{img}} =& - logD_{img}(I_{t-1}, I_t, I_{t+1}) \\ & - log(1 - D_{img}(I_{g_{t-1}}, I_{g_t}, I_{g_{t+1}})). \end{aligned} \tag{5.17}$$

To summarize, the full loss of G is given by

$$\begin{aligned} L_G =& \lambda_2 L_2 + \lambda_{uv,s} L_s^{uv} + \lambda_{smo} L_{smo} \\ & + \lambda_{uv,t} L_t^{uv} + \lambda_{img,s} L_s^{img} + \lambda_{img,t} L_t^{img}. \end{aligned} \tag{5.18}$$

In practice, we found it difficult to keep the losses in image space stable at the beginning of the training. Hence, we train G first with the partial loss L_{G_1}, where

$$L_{G_1} = \lambda_2 L_2 + \lambda_{uv,s} L_s^{uv} + \lambda_{smo} L_{smo} + \lambda_{uv,t} L_t^{uv}, \tag{5.19}$$

for 5×10^4 steps. We freeze G, and only train D_{img} for 5×10^4 steps to ensure that D_{img} is commensurate with G. We then train all networks jointly with the full loss L_G for another 10×10^4 steps. Generator G is built with ResNet architecture, using 30 (for DensePose P_t^r) or 20 (for SMPL P_t^r) residual blocks. All of our discriminators D_s, D_t, and D_{img} follow the same encoder structure using 5 convolutional layers followed by a dense layer. Details of our network architectures can be found in Appendix C. We use 120 continuous frames without background from the Fashion dataset [120] as the training data. For every step, we randomly crop small regions of size 32×32 from P_t^r to be used as input. The Adam optimizer is applied for training. Other learning details are given in the Supplementary.

Model inference. After the training, UV coordinates P_t^g with full clothing silhouettes can be generated via G. We can achieve pose-guided generation when a sequence of raw target poses is provided. Since we focus on the UV coordinates and inputs to G, which do not include texture information, virtual try-on can also be easily achieved in our pipeline by changing texture maps to which the UV coordinates are applied. Once P_t^g is generated, the image sequence I_t' requires a minimal number of calculations to be produced (essentially, only one texture lookup per output pixel). As we will demonstrate below, this is vastly more efficient than, e.g., evaluating a full CNN.

5.3 Ablation study

This section shows how different parts of Equation 5.18 influence the generated results. We start with a basic model trained with the losses L_2 and L_s^{uv} and denote this V_1. We then add temporal losses in the UV space, L_{smo} and L_t^{uv}, for training and denote this as V_2. The full model trained with L_G is denoted as V_3.

	PSNR↑	LPIPS↓ $\times 10^{-2}$	tOF↓ $\times 10^{4}$	tLP↓ $\times 10^{-2}$	T-diff↓ $\times 10^{5}$
P_t^r	22.1	8.1	1.69	1.0	5.42
V_1	**23.1**	7.9	1.84	1.4	**3.93**
V_2	**23.1**	**7.6**	1.70	**0.9**	4.33
V_3	22.9	7.7	**1.65**	1.0	4.19

Table 5.1: Quantitative comparisons between P_t^r and our three different versions, V_1, V_2, and V_3. For a fair comparison, the body shapes of V_1, V_2, and V_3 are cropped to be in line with P_t^r. Our method shows significant improvements on both spatial (PSNR and LPIPS) and temporal (tOF, T-diff) evaluation metrics.

Figure 5.5 shows two qualitative comparisons. All three versions successfully fill in the missing parts of the DensePose UV map and are close to the reference (green patch, skirt edge). To evaluate the temporal coherence, we zoom in on the motion of the flower patterns (blue patch). Results from the raw UV coordinates P^r are unsteady since its temporally unstable UV content leads to a misalignment of the texture over time. V_1 has better coherence due to the UV relocation (Sec. 5.2.3) applied to the training data. V_2 and V_3 show progressive improvements thanks to the temporal stability losses and the image space losses.

As quantitative evaluation of the spatial performance, we compute peak signal-to-noise ratio (PSNR) and perceptual LPIPS [147]. For temporal stability, we follow [99] and estimate the differences of warped frames, i.e., T-diff $= \|I_{g_t}, \mathcal{W}(I_{g_t}, v_t)\|_1$, where v_t typically denotes the intra-frame motion computed by optical flow. In our setting we use the UV coordinates for v_t instead (details in the Supplementary Material). Additionally, we evaluate with two temporal coherence metrics [200]: $tOF : \left\|OF(I_t, I_{t+1}) - OF(I_{g_t}, I_{g_{t+1}})\right\|_1$ and $tLP : \left\|LPIPS(I_t, I_{t+1}) - LPIPS(I_{g_t}, I_{g_{t+1}})\right\|_1$. Except for PSNR, lower values are better for all metrics.

From Table C.1, we see that that V_1 has the worst results in terms of tOF and tLP. Its LPIPS is also worse than V_2 and V_3 because V_1 is trained purely with spatial losses in the UV space. Hence, supervision via preprocessed data P_t^f is insufficient. Note, however, that V_1 shows the best T-diff score, as T-diff mainly relies on the calculation of v_t and is easily "fooled" by overly smooth content. V_2 and V_3 add temporal constraints and show better temporal behaviour in terms of tOF and tLP. Compared with V_2, V_3 exhibits a similar spatial performance though it yields better temporal stability. This is especially the case if we evaluate without cropping to fit P_t^r (see Supplementary). This also verifies that loss functions from the image space can be successfully applied to guide the training.

Optimized UVs (P_t^o). In addition to Figure 5.3c in Sec. 5.2.2, more samples of the optimized UVs P_t^o are shown in Figure 5.6 and the Supplementary. The comparison of PSNR and LPIPS scores in Figure 5.6 verifies that our optimization pipeline significantly improves the spatial content. Similar conclusions can be drawn for the UV coordinates derived from SMPL (see Figure 5.7).

		example in Figure 4	a)	b)	c)
PSNR ↑	$P^r(I_t)$	26.44	26.10	14.39	15.33
	$P^o(I_t)$	**36.40**	**40.92**	**36.91**	**30.11**
LPIPS↓ ($\times 10^{-1}$)	$P^r(I_t)$	0.65	1.09	1.30	1.38
	$P^o(I_t)$	**0.12**	**0.09**	**0.09**	**0.24**

Figure 5.6: Comparisons between DensePose UVs P_t^r and optimized UVs P_t^o. Here, we only show examples of the skirt part in T_t to clarify the differences. We can see that P_t^o can preserve most of the skirt information in T_t, and I_t' of P_t^o are closer to I_t than that of P_t^r. The quantitative evaluation also shows that our results after UV optimization (described in Sec. 5.2.2) are closer to the reference.

		a)	b)	c)
PSNR ↑	$P^r(I_t)$	16.61	13.33	19.68
	$P^o(I_t)$	**31.64**	**33.27**	**33.88**
LPIPS↓ ($\times 10^{-1}$)	$P^r(I_t)$	1.25	2.18	1.41
	$P^o(I_t)$	**0.10**	**0.43**	**0.21**

Figure 5.7: Comparisons between P_t^r from SMPL and P_t^o. We can see that after optimization, P_t^o preserves more of the loose clothing and I'_t closely matches I_t. Quantitative evaluations also show that our results are much closer to the reference.

5.4 Results and evaluation

Direct comparison of P_t^g. We provide a direct comparison between raw UVs P_t^r and those generated by our approach P_t^g in Figure 5.8. Apart from the body itself, it is visible that our outputs I'_t, generated with UVs from both SMPL and DensePose models, also recover the hair, sleeves, and the skirt. Hence, we have fulfilled the goal of capturing the full appearance of a person, rather than the body silhouette.

Comparison with state of the art. In Figure 5.9, we compare with the closest method DwNet, which also focuses on video generation from a single image with UV coordinates. DwNet smoothes the texture of the clothing as the quality of its output is limited by the accuracy of the warping module. However, our method focuses on UV coordinates, and obtains the appearance information directly from the texture map, so our results are significantly sharper. Our results are also closer to the reference images, leading to better spatial evaluations like PSNR and LPIPS. For temporal quality, the tOF and tLP values indicate that our results have better temporal stability than DwNet. We also conducted a user study to evaluate coherence (see Figure 5.10. Raw DensePose P_t^r is the baseline, while models V_1 and V_3 are trained with P_t^f, without and with temporal losses, respectively. V_3 gives significantly improved evaluations from the participants. We also

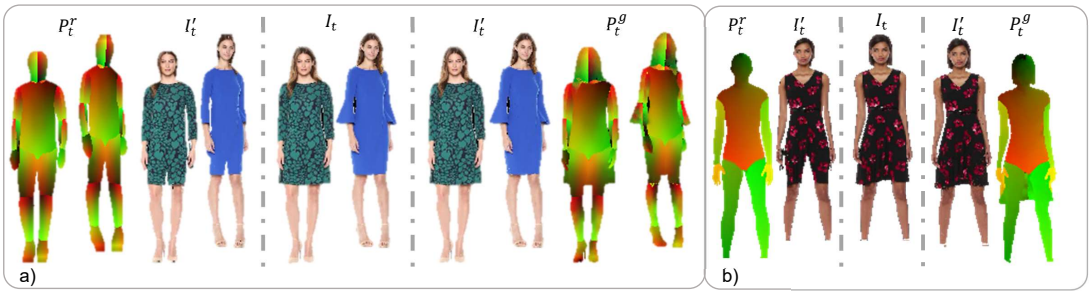
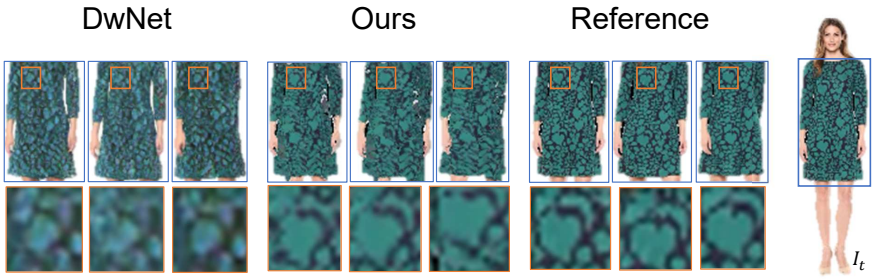


Figure 5.8: Comparison between P_t^g and P_t^r. P_t^g in (a) and (b) are generated from DensePose and SMPL model, respectively. Our I_t' are closer to the reference I_t, which indicates that P_t^g has better capacity to preserve more information of I_t.



	PSNR ↑	LPIPS ↓	tOF ↓ ($\times 10^4$)	tLP ↓ ($\times 10^{-2}$)	FPO (*times/pixel*)
DwNet	22.02	0.069	1.48	0.67	57.5M
Ours	**23.23**	**0.048**	**1.21**	**0.35**	**5**

Figure 5.9: Comparison with state-of-the-art method DwNet. Our results are closer to the reference, which is also supported by the evaluation metrics below. Additionally, we also compare the number of floating point operations (FPO) for every pixel during video generation. Without rerunning trained models, our method shows a significant reduction of computation.

outperform DwNet with high confidence, confirming the effectiveness of the temporal losses and the tOF and tLP evaluations.

We note that the DwNet model needs to be rerun once the texture is changed. In contrast, once the UV coordinates of a sequence have been generated, our method can re-texture a sequence without evaluating any trained models. Instead, we simply map the updated texture via our UV coordinates; this is a simple lookup that is several orders of magnitude fewer in operations than DwNet. Such a low computational load would, e.g., allow for running a virtual try-on pipeline in real-time on otherwise low-performance end-devices.

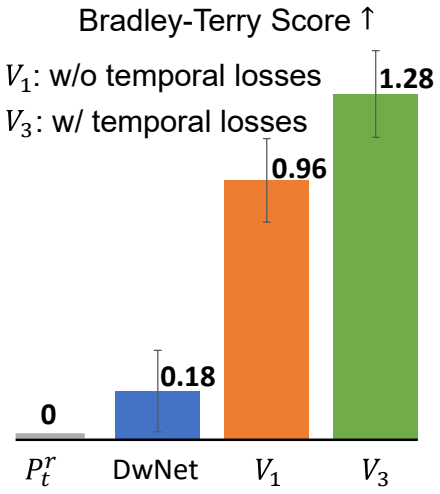


Figure 5.10: User study for the red-black dress case. Our full version V_3 significantly improves over V_1 and DwNet.

Generated video with different textures. Our generation network completely separates the UV representation from the RGB appearance information, which is only encoded in the constant texture T_o. As such, the UV coordinates generated from our model are compatible with any other texture that aligns with the arrangement of the original T_o. This makes it easy to create virtual try-on applications by modifying the texture. In particular, the source clothing can be obtained from any image source, e.g., another photo or a texture image. We show re-textured examples in Figure 5.1, Figure 5.12 and the Supplementary. Note that as our focus is on capturing clothing, hence we reuse the texture of the human parts (face, hands and legs) from the source videos for these virtual try-on results. Our pipeline can also be applied to datasets containing more diverse motions and complex backgrounds, such as the Tai-Chi dataset (see results in Figure 5.11a). Results are in line with the conclusions of our previous results: our optimization result P_t^o successfully recovers the missing UV coordinates and generates full images I_t'. After training, our synthesized result $(P_t^g + T_o)$ is closer to the reference than DensePose $(P_t^r + T_o)$ for temporal and spatial evaluations. Lastly, our models are specific to garment silhouettes, not individual videos. Retraining is only necessary if the silhouette changes. E.g. in Figure 5.11b, the model is trained with the sequence B (with sleeveless dress) and can be conditioned on poses in A (with long sleeves) to generate C. We aim for this direction since the silhouettes of common clothes are limited.

5.5 Conclusions

Limitations. Our method generates an entire video via P_t^g and T_o. Currently, we simply choose T_0 for T_o. However, T_0 may not have sufficient coverage for some situations, e.g., the backside of the clothing. This could be improved by incorporating additional

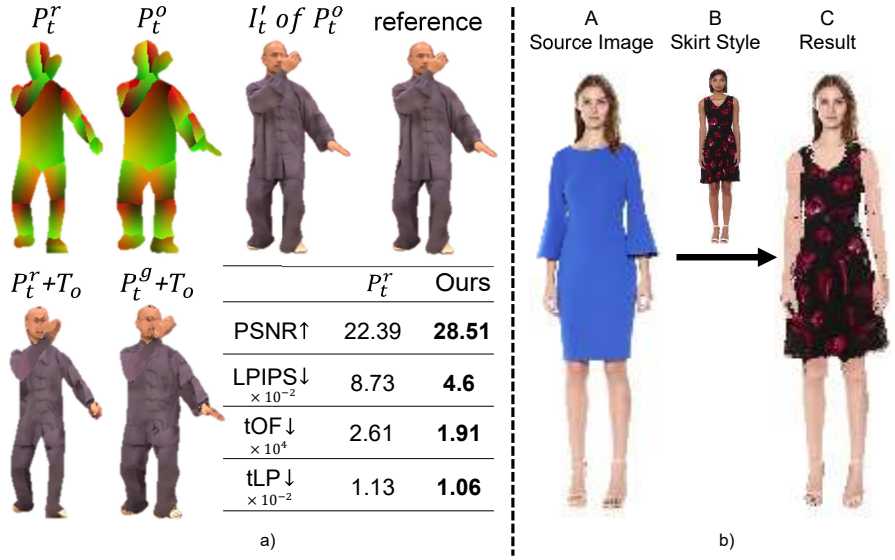


Figure 5.11: a) Results of the Tai-Chi dataset. b) Pose-guided generation application. Our model is generalized to different poses from different videos.

steps for texture completion [201, 202]. RevisionAnother limitation arises from boundary occlusions. While we aim at coherent point correspondence among different frames, occlusions occurred in the boundary areas make it impossible to find the obscured point at frame $t+1$ for the corresponding point at frame t, which brings a noticeable degree of high-frequency noise near the boundaries during fast motions of the body or clothing. But quantitative metrics and our user study show that our results yield better temporal coherence than state-of-the-art methods. Besides, this problem could benefit from additional image space smoothing over time.

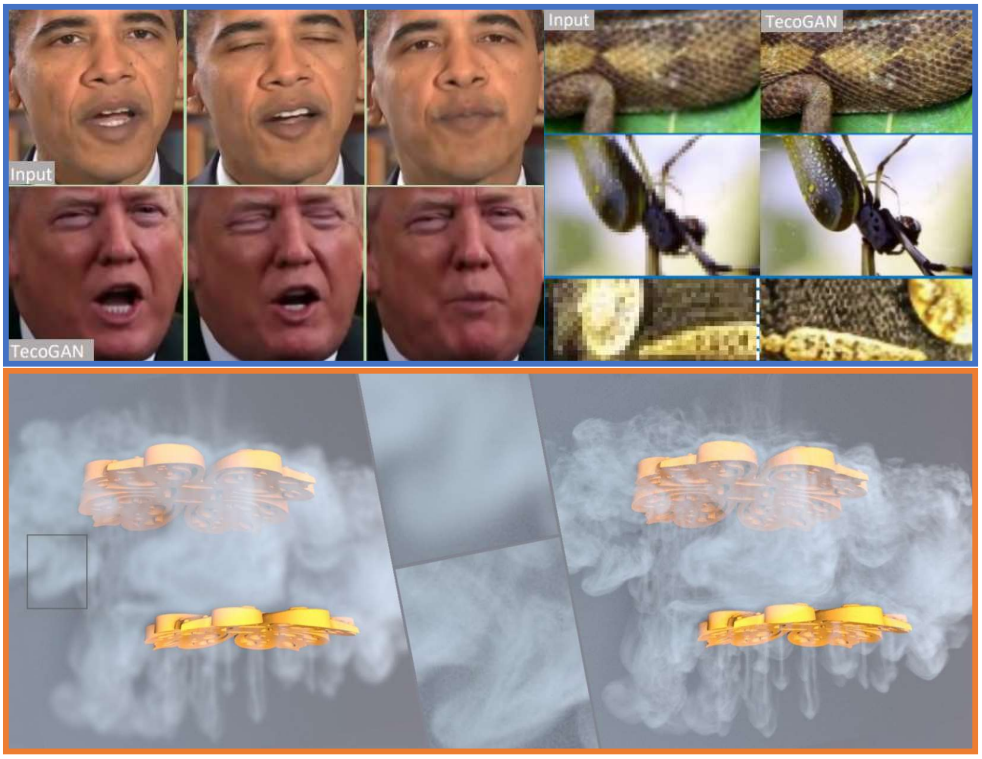
Conclusion. We have presented a novel algorithm to generate stable UV coordinates for image sequences that capture the full appearance of a human body, including loose clothing and hair. Central in arriving at this goal are a custom pre-computation pipeline and a spatio-temporal adversarial learning approach. Our method allows for high-quality video generation and also enables very quick turnaround times for style modifications. Based on the one-time process to generate a UV coordinate sequence, our method allows for the repeated synthesis of output videos via a single underlying texture with vastly reduced computations compared to existing approaches. RevisionCurrently, we primarily focus on clothing, because the complicated textures and various poses of the body make it a challenging application. It provides an appropriate test bed that encapsulates the capabilities of our pipeline. However, our pipeline can be potentially generalized to UVs

of other objects, e.g., animals, cars and furniture. This enables us to achieve video generation and texture editing of arbitrary objects easily in our future work.

Figure 5.12: P_t^g is compatible with different textures to generate a desired target sequence. (a)-(c) are generated using DensePose UV coordinates, while (d) uses SMPL UVs.

Part IV

Conclusion of Temporally Coherent Video Generation



"Wherever you go, go with all your heart."

Confucius

6 Temporally Coherent Video Generation

Keeping generated sequences temporally coherent is a crucial goal of video generation. In Part II, we have presented a temporally coherent SR algorithm for fluid flow. We proposed a novel temporal discriminator to improve the temporal coherence of the generated flow sequence. Besides, we proposed a novel regularizer to improve the generalization of trained models, then pre-trained models can be efficiently reused for related tasks. In Part III, we achieved differentiable transformations among RGB image, texture, and UV coordinate maps. TemporalUV algorithm was introduced to generate temporal coherent

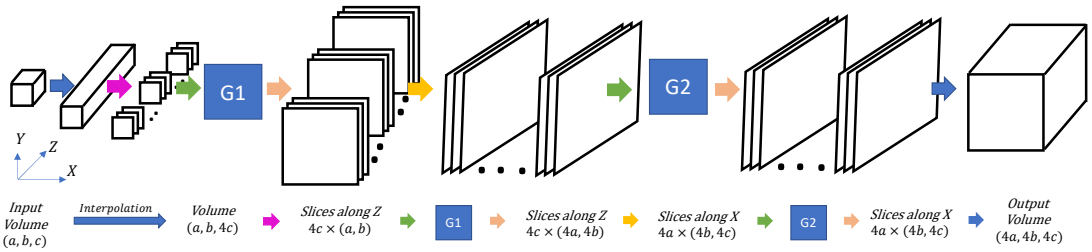


Figure 6.1: The pipeline of the proposed multi-pass GAN. After an up-sampling along z, we process two orthogonal directions with two adversarially trained generator networks G_1 and G_2. The initial up-sampling ensures that all unknowns are processed evenly by the networks.

UV coordinate maps. In this chapter, we will continue the discussion in Part II and Part III, and address different sequence generation problems with GANs.

In Part II, we achieved SR of the fluid flow with a scaling factor of four, which is hard to be increased under the architecture of tempoGAN because of the large volume data size. We propose a novel method to up-sample volumetric functions with GANs using several orthogonal passes. Our method decomposes generative problems on Cartesian field functions into multiple smaller sub-problems that can be learned more efficiently. Specifically, as shown in Figure 6.1, we utilize two separate generative adversarial networks: the first one up-scales slices which are parallel to the XY-plane, whereas the second one refines the whole volume along the Zaxis working on slices in the YZ-plane. In this way, we obtain full coverage for the 3D target function and can leverage spatio-temporal supervision with a set of discriminators. Additionally, we demonstrate that our method can be combined with curriculum learning and progressive growing approaches. We arrive at a first method that can up-sample volumes by a factor of eight along each dimension, i.e., increasing the number of degrees of freedom by 512. Large volumetric up-scaling factors such as this one have previously not been attainable as the required number of weights in the neural networks renders adversarial training runs prohibitively difficult. Despite making it possible to achieve large up-scaling factors, our method also reduces training time. Training our 4× multi-pass network took approximately 3 days for the first generator and 2 days for the second. This is almost twice as fast as training a 3D model reported in previous work [203]. We additionally only employed a single GTX 1080 Ti instead of two GPUs. Training the progressively growing network for an 8× up-scaling took about 8 and 5 days for the first and second generator network, respectively. As this scale is not feasible with previous work, we cannot compare training times for the 8× case. Similar to previous work, the memory available in current GPUs can be a bottleneck when applying the trained networks to new input and can make it necessary to subdivide the inputs into tiles. Regarding our implementation, we can deal with full slices of up to approximately 256^3, i.e., do not need to apply tiling. Example results are shown in the teaser image (the bottom part) of this chapter.

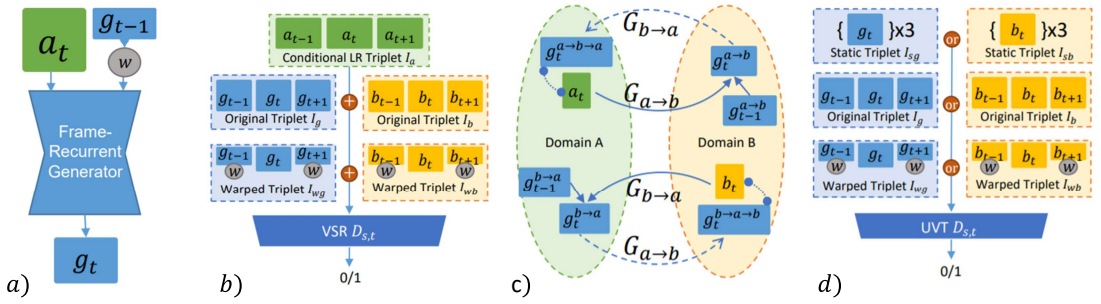


Figure 6.2: a) The frame-recurrent VSR Generator. b) Conditional VSR $D_{s,t}$. c) The UVT cycle link formed by two recurrent generators. d) Unconditional UVT $D_{s,t}$.

Except for fluid flow, temporal self-supervision is also applied to GAN-based video-to-video transformation, such as video SR and unpaired video translation. For the former, state-of-the-art methods often favor simpler norm losses such as L^2 over adversarial training. However, their averaging nature easily leads to temporally smooth results with an undesirable lack of spatial detail. For unpaired video translation, existing approaches modify the generator networks to form spatio-temporal cycle consistencies. In our work, we propose a novel adversarial learning method for a recurrent training approach that supervises both spatial contents as well as temporal relationships, as shown in Figure 6.2. With no ground truth motion available, the spatio-temporal adversarial loss and the recurrent structure enable our model to generate realistic results while keeping the generated structures coherent over time. For both tasks, we show that temporal adversarial learning is key to achieving temporally coherent solutions without sacrificing spatial detail. In addition to the adversarial network which supervises the short-term temporal coherence, long-term consistency is self-supervised using a novel bi-directional loss formulation, as shown in Figure 6.3, which we refer to as "Ping-Pong" (PP) loss in the following. The PP loss effectively avoids the temporal accumulation of artifacts without depressing detailed features, which can potentially benefit a variety of recurrent architectures. Additionally, We also note that most existing image metrics focus on spatial content only. We propose a first set of metrics to quantitatively evaluate the accuracy as well as the perceptual quality of the temporal evolution, which measures the perceptual similarity over time and the similarity of motions with respect to a ground truth reference. A series of user studies confirm the rankings computed with these metrics. Example results are shown in the teaser image (the upper part) of this chapter.

In this chapter, we discussed more generative tasks and discussed our solutions with GANs. Explorations in different tasks and datasets show us the capability of GANs to generate temporally coherent sequences with promising details. In the next chapter, we will summarize all the methods illustrated in this dissertation, present our experience of video generation with GANs, and discuss the limitations and potential future directions of our work.

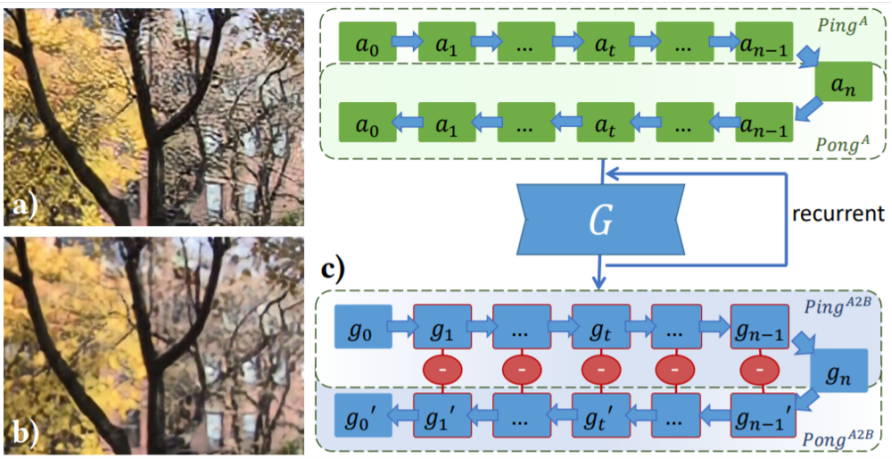


Figure 6.3: a) Result without PP loss. The VSR network is trained with a recurrent frame-length of 10. When inference on long sequences, frame 15 and latter frames of the foliage scene show the drifting artifacts. b) Result trained with PP loss. These artifacts are removed successfully for the latter. c) When inferring a symmetric PP sequence with a forward pass (Ping) and its backward counterpart (Pong), our PP loss constrains the output sequence to be symmetric. It reduces the L^2 distance between g_t and g'_t, the corresponding frames in the forward and backward passes, shown via red circles with a minus sign. The PP loss reduces drifting artifacts and improves temporal coherence.

"The future belongs to those who prepare for it today."
Malcolm X

7 Conclusion and Future Work

In this dissertation, we primarily focus on temporally coherent video generation with GANs, including fluid flow and natural video generation. In this chapter, we will start with a summary of all the methods introduced in previous chapters. Then, we will discuss the limitation of those methods and potential future directions.

7.1 Method Summary

In previous chapters, we introduced our GAN-based sequence generation algorithms among different scenarios, such as fluid flow, and natural videos. Discriminator from a standard GAN receives one single image as input, which purely provides spatial feedback for the generator. However, tiny changes in the input can bring significant differences in the output. Thus, temporal coherence of the results cannot be constrained via normal GANs. Instead, we proposed spatio-temporal adversarial training, and used multiple temporally adjacent frames as input of the discriminator, then feedback of the temporal dimension can be transferred from the discriminator to the generator.

In Figure 7.1, we summarized the connections among the proposed algorithms, which are repeated from Figure 1.3. Practical experience is summarized as follows. Standard GAN training is good at generation problems with multi-modality, such as image SR, and results yield high perceptual quality. But temporal coherence of the results is still out of consideration. Low-level temporal losses, such as L^2, can bring limited improvements in the temporal dimension. Our proposed spatio-temporal adversarial methods offer

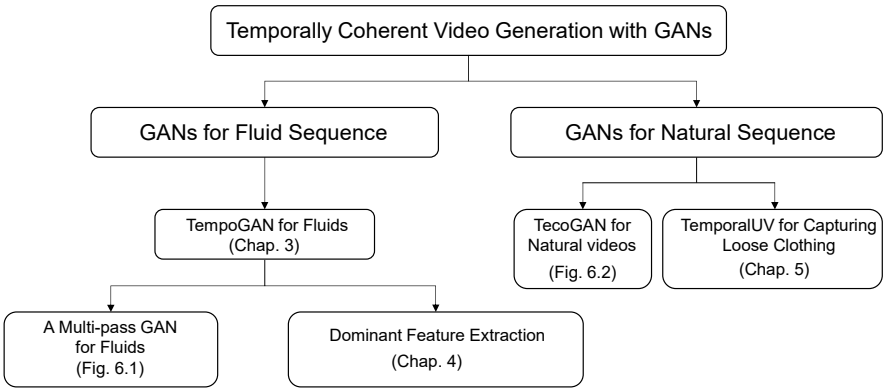


Figure 7.1: Connections between different algorithms.

promising coherent sequences with realistic details. On the other hand, we find that motion compensation helps improve the temporal coherence, e.g., the velocity of the fluid flow and optical flow of the natural sequence. Besides, the recurrent structure can reuse information from the previous results, which simplifies the generative problem and yields better temporal coherence. However, artifacts in the previously generated frames may also be amplified in the later frames, which can be improved with our proposed Ping-Pong loss. Furthermore, for complicated problems like sequence generation, curriculum learning takes the strategy to break down the problem into smaller sub-problems, which allows us to start from a simpler sub-problem and increase the complexity step by step. In our case, we start from a low scaling factor for the SR task and increase the scaling factor gradually. For GAN training, keeping both generator and discriminator at a balanced level is critical. We find that the dynamic training strategy is helpful, i.e., stopping the training of the discriminator when the discriminator is stronger than the generator, and restarting the training again when the generator becomes stronger.

7.2 Limitations and Future Work

In this dissertation, we introduced GAN-based sequence generation algorithms for various tasks in computer graphics and vision. However, as mentioned in previous chapters, our proposed algorithms exhibit specific limitations in specific scenarios.

GANs for Fluid Flow Our fluid flow SR algorithms did not consider the physical relationship between low- and high- resolution data. According to the physical formulation, the energy dissipation rate will be changed under different resolutions, i.e., even with the same initial parameters, the shape of the flow should be changed for simulations with different resolutions. However, in our fluid flow SR algorithms, we purely increase the resolution while preserving the shape of the flow. In this case, our algorithms are good at designing the required HR flow shape, since we can efficiently adjust the flow

shape at LR before increasing the resolution. On the other hand, our algorithms only considered the density and velocity as the inputs of the generator, which can achieve limited control of the results.

GANs for Natural Videos Our temporalUV algorithm achieved temporal coherent UV coordinate maps generation. However, in this algorithm, we purely synthesize the foreground part, i.e., the human body, and the background is out of consideration. On the other hand, lighting condition is also not part of the model. Thus, it would be hard to adjust the lighting in the results with our model.

7.3 Future Work

From our proposed algorithms and results, we can see the power of deep learning methods for sequence generation. These directions are worth further exploring in the future.

Fluid Flow Generation To correct the physical rules in the generated results, one possible solution is to map dynamic parameters to the density outputs directly, such as [35, 204]. Then outputs can be controlled via the input parameters. However, physical constraints, e.g., boundary conditions, still cannot be applied during training. On the other hand, we could apply physical constraints in the loss functions, such as [205], or apply a differentiable PDE solver [206] as part of our pipeline. Currently, we only focus on single-phase fluid flow generation. However, multi-phase fluid flow is more common in our daily life. Thus, applying our algorithms for multi-phase fluid flow generation would also be an interesting direction.

Except for fluid data generation, result rendering is another step that consumes most of the time and computational resources. Deep learning, as a powerful tool, has also been applied in image rendering, such as NeRF [207]. However, NeRF primarily works on solid static objects. We think it would be interesting to achieve fluid volume data rendering with deep learning methods.

Pose-Guided Human Video Generation Currently, our temporalUV algorithm purely used 2D RGB images as source images. In the future, we could also take the depth information as part of our inputs, since depth cameras are now common setups in the customer products, e.g., mobile phones. With the depth information, the foreground and background can be easily separated. Then, full background can also be synthesized, such as [126]. Besides, lighting effects can also be beneficial from the depth information and make it possible to adjust the lighting effects for the foreground and background objects.

Now we only apply our algorithm with human video generation, since we think this is a tough task with complicated motions. However, UV coordinate maps are widely used in various applications. In the future, we will also extend our algorithm for other objects, such as cars, animals, furniture, etc.

In summary, sequence generation is an important application, which is closely connected to our daily life. Our algorithms and results demonstrate that deep learning methods are powerful in dealing with sequence generation problems and yield state-of-the-art results. Applying deep learning methods for sequence generation is worth further exploration. Beyond tasks we discussed above, we hope that the works introduced in our dissertation can provide some insights and inspiration for other related tasks, such as frame interpolation, video enhancement, and video editing.

Bibliography

[1] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[2] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, 2000.

[3] M. Szummer and R. W. Picard. Temporal texture modeling. In *Proceedings of 3rd IEEE International Conference on Image Processing*, volume 3, pages 823–826. IEEE, 1996.

[4] J. Xie, S.-C. Zhu, and Y. Nian Wu. Synthesizing dynamic patterns by spatial-temporal generative convnet. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 7093–7101, 2017.

[5] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.

[6] T.-H. Wang, Y.-C. Cheng, C. H. Lin, H.-T. Chen, and M. Sun. Point-to-point video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10491–10500, 2019.

[7] J. He, A. Lehrmann, J. Marino, G. Mori, and L. Sigal. Probabilistic video generation using holistic attribute control. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 452–467, 2018.

[8] J. Pan, C. Wang, X. Jia, J. Shao, L. Sheng, J. Yan, and X. Wang. Video generation from single semantic label map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2019.

[9] J. Walker, A. Razavi, and A. v. d. Oord. Predicting video with vqvae. *arXiv preprint arXiv:2103.01950*, 2021.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[11] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

[12] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

[13] N. Thürey, C. Wojtan, M. Gross, and G. Turk. A multiscale approach to mesh-based surface tension flows. *ACM Transactions on Graphics (TOG)*, 29(4):1–10, 2010.

[14] D. Roetenberg, H. Luinge, and P. Slycke. Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, 1, 2009.

[15] B. Nguyen. Developing a game debugger with unreal engine 4. 2021.

[16] R. Alp Güler, N. Neverova, and I. Kokkinos. Densepose: Dense human pose estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7297–7306, 2018.

[17] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, 1990.

[18] J. Stam. Stable fluids. In *Siggraph*, volume 99, pages 121–128, 1999.

[19] B. Kim, Y. Liu, I. Llamas, and J. R. Rossignac. Flowfixer: Using bfecc for fluid simulation. Technical report, Georgia Institute of Technology, 2005.

[20] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 2008.

[21] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. In *ACM Transactions on Graphics (TOG)*, volume 26, page 100. ACM, 2007.

[22] Y. Teng, D. I. Levin, and T. Kim. Eulerian solid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 35(6):200, 2016.

[23] T. Kim, N. Thürey, D. James, and M. Gross. Wavelet turbulence for fluid simulation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 50. ACM, 2008.

[24] R. Narain, J. Sewall, M. Carlson, and M. C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 166. ACM, 2008.

[25] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

[26] J. Zehnder, R. Narain, and B. Thomaszewski. An advection-reflection solver for detail-preserving fluid simulation. *ACM Transactions on Graphics (TOG)*, 37(4):1–8, 2018.

[27] Z. Qu, X. Zhang, M. Gao, C. Jiang, and B. Chen. Efficient and conservative fluids using bidirectional mapping. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[28] G.-H. Cottet, P. D. Koumoutsakos, et al. *Vortex methods: theory and practice*, volume 8. Cambridge university press Cambridge, 2000.

[29] S. Weißmann and U. Pinkall. Filament-based smoke with vortex shedding and variational reconnection. In *ACM SIGGRAPH 2010 papers*, pages 1–12. 2010.

[30] T. Pfaff, N. Thuerey, and M. Gross. Lagrangian vortex sheets for animating fluids. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

[31] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, 2001.

[32] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. In *ACM SIGGRAPH 2005 Papers*, pages 910–914. 2005.

[33] T. Pfaff, N. Thuerey, J. Cohen, S. Tariq, and M. Gross. Scalable fluid simulation using anisotropic turbulence particles. In *ACM SIGGRAPH Asia 2010 papers*, pages 1–8. 2010.

[34] S. C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM Journal on Scientific and Statistical Computing*, 2(1):1–4, 1981.

[35] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[36] S. Wiewel, M. Becher, and N. Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

[37] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

[38] R. Ando, N. Thürey, and C. Wojtan. A dimension-reduced pressure solver for liquid simulations. In *Computer Graphics Forum*, volume 34, pages 473–480. Wiley Online Library, 2015.

[39] M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Transactions on Graphics (TOG)*, 29(4):1–9, 2010.

[40] M. Aanjaneya, M. Gao, H. Liu, C. Batty, and E. Sifakis. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

[41] K. Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3):1–22, 2013.

[42] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In *Symposium on Computer Animation*, pages 65–73, 2010.

[43] M. Becker and M. Teschner. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217, 2007.

[44] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. Implicit incompressible sph. *IEEE transactions on visualization and computer graphics*, 20(3):426–435, 2013.

[45] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58(5):471–483, 1996.

[46] J. U. Brackbill, D. B. Kothe, and H. M. Ruppel. Flip: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38, 1988.

[47] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.

[48] S. Markidis, V. Olshevsky, C. P. Sishtla, S. W. Chien, E. Laure, and G. Lapenta. Polypic: the polymorphic-particle-in-cell method for fluid-kinetic coupling. *Frontiers in Physics*, 6:100, 2018.

[49] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[50] C. Jiang, T. Gast, and J. Teran. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[51] A. P. Tampubolon, T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Museth. Multi-species simulation of porous sand and water mixtures. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. NIPS, 2012.

[53] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. pages 580–587. IEEE, 2014.

[54] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. *arXiv preprint arXiv:1703.07511*, 2017.

[55] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5515–5524, 2016.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[57] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *stat*, 1050:10, 2014.

[58] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. arxiv 2017. *arXiv preprint arXiv:1701.07875*, 30:4, 2017.

[59] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[60] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[61] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[62] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[63] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Stabilizing training of generative adversarial networks through regularization. *Advances in neural information processing systems*, 30, 2017.

[64] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[65] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR*, 2016.

[66] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv:1703.10593*, 2017.

[67] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[68] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. 2017.

[69] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv:1609.04802*, 2016.

[70] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. NováK, A. Harvill, P. Sen, T. Derose, and F. Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Transactions on Graphics (TOG)*, 36(4):97, 2017.

[71] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017.

[72] S. Kallweit, T. Müller, B. McWilliams, M. Gross, and J. Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *arXiv:1709.05418*, 2017.

[73] L. Mosser, O. Dubrule, and M. J. Blunt. Reconstruction of three-dimensional porous media using generative adversarial neural networks. *arXiv:1704.03225*, 2017.

[74] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[75] A. B. Farimani, J. Gomes, and V. S. Pande. Deep learning the physics of transport phenomena. *arXiv preprint arXiv:1709.02432*, 2017.

[76] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

[77] L. Ladicky, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. 34(6):199, 2015.

[78] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. *arXiv: 1607.03597*, 2016.

[79] L. Prantl, B. Bonev, and N. Thuerey. Pre-computed liquid spaces with generative neural networks and optical flow. *arXiv:1704.07854*, 2017.

[80] L. Prantl, B. Bonev, and N. Thuerey. Pre-computed liquid spaces with generative neural networks and optical flow. *arXiv preprint arXiv:1704.07854*, 2017.

[81] K. Um, X. Hu, and N. Thuerey. Liquid splash modeling with neural networks. In *Computer Graphics Forum*, volume 37, pages 171–182. Wiley Online Library, 2018.

[82] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *arXiv preprint arXiv:1806.02071*, 2018.

[83] M. Chu and N. Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. 36(4)(69), 2017.

[84] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[85] M. He, D. Chen, J. Liao, P. V. Sander, and L. Yuan. Deep exemplar-based colorization. *ACM Transactions on Graphics (TOG)*, 37(4):1–16, 2018.

[86] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[87] O. Jamriska, S. Sochorová, O. Texler, M. Lukác, J. Fiser, J. Lu, E. Shechtman, and D. Sỳkora. Stylizing video by example. *ACM Transactions on Graphics (TOG)*, 38(4):1–11, 2019.

[88] V. Sitzmann, S. Diamond, Y. Peng, X. Dun, S. Boyd, W. Heidrich, F. Heide, and G. Wetzstein. End-to-end optimization of optics and image processing for achromatic extended depth of field and super-resolution imaging. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[89] B. Wronski, I. Garcia-Dorado, M. Ernst, D. Kelly, M. Krainin, C.-K. Liang, M. Levoy, and P. Milanfar. Handheld multi-frame super-resolution. *ACM Transactions on Graphics (TOG)*, 38(4):1–18, 2019.

[90] B. Zhang, M. He, J. Liao, P. V. Sander, L. Yuan, A. Bermak, and D. Chen. Deep exemplar-based video colorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8052–8061, 2019.

[91] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[92] M. Haris, G. Shakhnarovich, and N. Ukita. Recurrent back-projection network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3897–3906, 2019.

[93] Y. Jo, S. W. Oh, J. Kang, and S. J. Kim. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3224–3232, 2018.

[94] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, and T. Huang. Robust video super-resolution with learned temporal dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2507–2515, 2017.

[95] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia. Detail-revealing deep video super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4472–4480, 2017.

[96] M. Saito, E. Matsumoto, and S. Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision*, pages 2830–2839, 2017.

[97] M. S. Sajjadi, R. Vemulapalli, and M. Brown. Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6626–6634, 2018.

[98] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[99] D. Chen, J. Liao, L. Yuan, N. Yu, and G. Hua. Coherent online video style transfer. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1105–1114, 2017.

[100] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos. In *German conference on pattern recognition*, pages 26–36. Springer, 2016.

[101] E. Pérez-Pellitero, M. S. Sajjadi, M. Hirsch, and B. Schölkopf. Photorealistic video super resolution. *arXiv preprint arXiv:1807.07930*, 2:4, 2018.

[102] K. Park, S. Woo, D. Kim, D. Cho, and I. S. Kweon. Preserving semantic and temporal consistency for unpaired video-to-video translation. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1248–1257, 2019.

[103] P. Bhattacharjee and S. Das. Temporal coherency based criteria for predicting video frames using deep multi-stage generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 4271–4280, 2017.

[104] D. Liu, Z. Wang, Y. Fan, X. Liu, Z. Wang, S. Chang, and T. Huang. Robust video super-resolution with learned temporal dynamics. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[105] C. Yang, Z. Wang, X. Zhu, C. Huang, J. Shi, and D. Lin. Pose guided human video generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 201–216, 2018.

[106] H. Tang, S. Bai, L. Zhang, P. H. Torr, and N. Sebe. Xinggan for person image generation. In *European Conference on Computer Vision*, pages 717–734. Springer, 2020.

[107] A. Grigorev, A. Sevastopolsky, A. Vakhitov, and V. Lempitsky. Coordinate-based texture inpainting for pose-guided image generation. *arXiv preprint arXiv:1811.11459*, 2018.

[108] W. Liu, Z. Piao, J. Min, W. Luo, L. Ma, and S. Gao. Liquid warping gan: A unified framework for human motion imitation, appearance transfer and novel view synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5904–5913, 2019.

[109] N. Neverova, R. Alp Guler, and I. Kokkinos. Dense pose transfer. In *Proceedings of the European conference on computer vision (ECCV)*, pages 123–138, 2018.

[110] G. Pavlakos, L. Zhu, X. Zhou, and K. Daniilidis. Learning to estimate 3d human pose and shape from a single color image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 459–468, 2018.

[111] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *European conference on computer vision*, pages 561–578. Springer, 2016.

[112] M. Madadi, H. Bertiche, and S. Escalera. Smplr: Deep smpl reverse for 3d human pose and shape recovery. *arXiv preprint arXiv:1812.10766*, 2018.

[113] M. Kocabas, N. Athanasiou, and M. J. Black. Vibe: Video inference for human body pose and shape estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5253–5263, 2020.

[114] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015.

[115] H. Yan, J. Chen, X. Zhang, S. Zhang, N. Jiao, X. Liang, and T. Zheng. Ultrapose: Synthesizing dense pose with 1 billion points by human-body decoupling 3d model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10891–10900, 2021.

[116] N. Neverova, J. Thewlis, R. A. Guler, I. Kokkinos, and A. Vedaldi. Slim densepose: Thrifty learning from sparse annotations and motion cues. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10915–10923, 2019.

[117] L. Ma, Z. Lin, C. Barnes, A. A. Efros, and J. Lu. Unselfie: Translating selfies to neutral-pose portraits in the wild. In *European Conference on Computer Vision*, pages 156–173. Springer, 2020.

[118] T. Zhu, P. Karlsson, and C. Bregler. Simpose: Effectively learning densepose and surface normals of people from simulated data. In *European Conference on Computer Vision*, pages 225–242. Springer, 2020.

[119] A. Pumarola, V. Goswami, F. Vicente, F. De la Torre, and F. Moreno-Noguer. Unsupervised image-to-video clothing transfer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[120] P. Zablotskaia, A. Siarohin, B. Zhao, and L. Sigal. Dwnet: Dense warp-based network for pose-guided human video generation. *arXiv preprint arXiv:1910.09139*, 2019.

[121] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool. Pose guided person image generation. In *Advances in neural information processing systems*, pages 406–416, 2017.

[122] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. Guttag. Synthesizing images of humans in unseen poses. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8340–8348, 2018.

[123] A. Siarohin, E. Sangineto, S. Lathuiliere, and N. Sebe. Deformable gans for pose-based human image generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3408–3416, 2018.

[124] A. Pumarola, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer. Unsupervised person image synthesis in arbitrary poses. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8620–8628, 2018.

[125] A. Siarohin, S. Lathuilière, S. Tulyakov, E. Ricci, and N. Sebe. Animating arbitrary objects via deep motion transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2377–2386, 2019.

[126] J. S. Yoon, L. Liu, V. Golyanik, K. Sarkar, H. S. Park, and C. Theobalt. Pose-guided human animation from a single image in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15039–15048, 2021.

[127] A. Siarohin, S. Lathuilière, S. Tulyakov, E. Ricci, and N. Sebe. First order motion model for image animation. In *Advances in Neural Information Processing Systems*, pages 7137–7147, 2019.

[128] K. Aberman, M. Shi, J. Liao, D. Lischinski, B. Chen, and D. Cohen-Or. Deep video-based performance cloning. In *Computer Graphics Forum*, volume 38, pages 219–233. Wiley Online Library, 2019.

[129] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5933–5942, 2019.

[130] K. Cheng, H.-Z. Huang, C. Yuan, L. Zhou, and W. Liu. Multi-frame content integration with a spatio-temporal attention mechanism for person video motion transfer. *arXiv preprint arXiv:1908.04013*, 2019.

[131] L. Liu, W. Xu, M. Zollhoefer, H. Kim, F. Bernard, M. Habermann, W. Wang, and C. Theobalt. Neural rendering and reenactment of human actor videos. *ACM Transactions on Graphics (TOG)*, 38(5):1–14, 2019.

[132] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Advances in neural information processing systems*, pages 613–621, 2016.

[133] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh. Recycle-gan: Unsupervised video retargeting. In *Proceedings of the European conference on computer vision (ECCV)*, pages 119–135, 2018.

[134] Y. Chen, Y. Pan, T. Yao, X. Tian, and T. Mei. Mocycle-gan: Unpaired video-to-video translation. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 647–655, 2019.

[135] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[136] Z. Wang, J. Chen, and S. C. Hoi. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3365–3387, 2020.

[137] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*, 2015.

[138] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems*, pages 658–666, 2016.

[139] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[140] D. Berthelot, T. Schumm, and L. Metz. BeGAN: Boundary equilibrium generative adversarial networks. *arXiv:1703.10717*, 2017.

[141] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[142] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, volume 1, page 3, 2017.

[143] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. URL: `http://distill.pub/2016/deconv-checkerboard`, `doi:10.23915/distill.00003`.

[144] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. 38(9):1734–1747, 2016.

[145] H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, et al. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

[146] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[147] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[148] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[149] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[150] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[151] D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pretraining help deep learning? In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 201–208, 2010.

[152] Y. Zhou and V. Govindaraju. Learning deep autoencoders without layer-wise training. *stat*, 1050:14, 2014.

[153] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[154] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, pages 177–185, 1989.

[155] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems*, pages 875–882, 1991.

[156] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[157] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[158] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[159] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[160] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[161] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[162] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[163] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[164] M. Ozay and T. Okatani. Optimization on submanifolds of convolution kernels in cnns. *arXiv preprint arXiv:1610.07008*, 2016.

[165] K. Jia, D. Tao, S. Gao, and X. Xu. Improving training of deep neural networks via singular value bounding. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4344–4352, 2017.

[166] N. Bansal, X. Chen, and Z. Wang. Can we gain more from orthogonality regularizations in training deep cnns? In *Advances in Neural Information Processing Systems*, pages 4266–4276. Curran Associates Inc., 2018.

[167] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, pages 2214–2224, 2017.

[168] J.-H. Jacobsen, A. Smeulders, and E. Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

[169] L. Zhang, Y. Lu, G. Song, and H. Zheng. Rc-cnn: Reverse connected convolutional neural network for accurate player detection. In *Pacific Rim International Conference on Artificial Intelligence*, pages 438–446. Springer, 2018.

[170] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[171] K. Gopalakrishnan, S. K. Khaitan, A. Choudhary, and A. Agrawal. Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials*, 157:322–330, 2017.

[172] H. Ding, S. K. Zhou, and R. Chellappa. Facenet2expnet: Regularizing a deep face recognition net for expression recognition. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 118–126. IEEE, 2017.

[173] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[174] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[175] M. E. Wall, A. Rechtsteiner, and L. M. Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[176] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[177] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[178] J. Walters-Williams and Y. Li. Estimation of mutual information: A survey. In *International Conference on Rough Sets and Knowledge Technology*, pages 389–396. Springer, 2009.

[179] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu. Orthogonal convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11505–11515, 2020.

[180] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.

[181] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, 2019.

[182] M. Alberti, M. Seuret, R. Ingold, and M. Liwicki. A pitfall of unsupervised pre-training. *arXiv preprint arXiv:1703.04332*, 2017.

[183] M.-L. Eckert, K. Um, and N. Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics (TOG)*, 38(6):239, 2019.

[184] Y. Zhou, D. Arpit, I. Nwogu, and V. Govindaraju. Is joint training better for deep auto-encoders? *arXiv preprint arXiv:1405.1380*, 2014.

[185] V. Eyring, S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor. Overview of the coupled model intercomparison project phase 6 (cmip6) experimental design and organization. *Geoscientific Model Development*, 9(5):1937–1958, 2016.

[186] S. Rasp and N. Thuerey. Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench. *Journal of Advances in Modeling Earth Systems*, page e2020MS002405, 2021.

[187] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[188] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[189] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[190] M. Du, N. Liu, and X. Hu. Techniques for interpretable machine learning. *arXiv preprint arXiv:1808.00033*, 2018.

[191] J. Deng, S. Cheng, N. Xue, Y. Zhou, and S. Zafeiriou. Uv-gan: Adversarial facial uv map completion for pose-invariant face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7093–7102, 2018.

[192] A. R. Farras, S. E. Guerrero, and M. Madadi. Rgb to 3d garment reconstruction using uv map representations. 2021.

[193] B. Gecer, J. Deng, and S. Zafeiriou. Ostec: One-shot texture completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7628–7638, 2021.

[194] Y. Wang, P. Bilinski, F. Bremond, and A. Dantcheva. Imaginator: Conditional spatio-temporal gan for video generation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1160–1169, 2020.

[195] R. Poranne, M. Tarini, S. Huber, D. Panozzo, and O. Sorkine-Hornung. Autocuts: simultaneous distortion and cut optimization for uv mapping. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017.

[196] J. D. Yang and S. Y. Shang. Cloth modeling simulation based on mass spring model. In *Applied Mechanics and Materials*, volume 310, pages 676–683. Trans Tech Publ, 2013.

[197] T. Liu, A. W. Bargteil, J. F. O'Brien, and L. Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[198] F. Theil. Surface energies in a two-dimensional mass-spring model for crystals. *ESAIM: Mathematical Modelling and Numerical Analysis*, 45(5):873–899, 2011.

[199] M. Belkin, P. Niyogi, and V. Sindhwani. On manifold regularization. In *International Workshop on Artificial Intelligence and Statistics*, pages 17–24. PMLR, 2005.

[200] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé, and N. Thuerey. Learning temporal coherence via self-supervision for gan-based video generation. *ACM Transactions on Graphics (TOG)*, 39(4):75–1, 2020.

[201] J. Chibane and G. Pons-Moll. Implicit feature networks for texture completion from partial 3d data. In *European Conference on Computer Vision*, pages 717–725. Springer, 2020.

[202] A. Grigorev, A. Sevastopolsky, A. Vakhitov, and V. Lempitsky. Coordinate-based texture inpainting for pose-guided human image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12135–12144, 2019.

[203] Y. Xie, E. Franz, M. Chu, and N. Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

[204] M. Chu, N. Thuerey, H.-P. Seidel, C. Theobalt, and R. Zayer. Learning meaningful controls for fluids. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021.

[205] Q. He and A. M. Tartakovsky. Physics-informed neural network method for forward and backward advection-dispersion equations. *Water Resources Research*, 57(7):e2020WR029479, 2021.

[206] P. Holl, V. Koltun, and N. Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[207] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[208] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *Symposium on geometry processing*, volume 6, pages 156–164, 2003.

[209] N. Thuerey and T. Pfaff. MantaFlow, 2018. *http://mantaflow.com*.

[210] S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey. Weatherbench: A benchmark dataset for data-driven weather forecasting. *arXiv preprint arXiv:2002.00469*, 2020.

A Appendix of Chapter 3

A.1 Details of Architectures

To clearly specify our networks, we use the following notation. Let in(resolution, channels), out(resolution, output) present input and output information; NI(output-resolution) represent nearest-neighbor interpolation; C(output-resolution, filter size, output-channels) denote a convolutional layer. Our resolutions and filter sizes are the same for every spacial dimension for both 2D and 3D. Resolutions of feature maps are reduced when strides >1. We use RB to represent our residual blocks, and use C_S for adding residuals in a RB. E.g., $RB_3 : [C_A, \text{ReLU}, C_B] + [C_S], \text{ReLU}$ means $[(input \rightarrow C_A \rightarrow \text{ReLU} \rightarrow C_B) + (input \rightarrow C_S)] \rightarrow \text{ReLU}$, where + denotes element-wise addition. BN denotes batch normalization, which is not used in the last layer of G, the first layer of D_t and the first layer of D_s [65]. In addition, — denotes concatenation of layer outputs along the channel dimension.

Architectures of G, D_s and D_t:

G:
$in(16, 4)$
$NI(64, 4)$
$RB_0 : [C_A(64, 5, 8), BN, \text{ReLU}, C_B(64, 5, 32), BN] + [C_S(64, 1, 32), BN], \text{ReLU}$
$RB_1 : [C_A(64, 5, 128), BN, \text{ReLU}, C_B(64, 5, 128), BN] + [C_S(64, 1, 128), BN], \text{ReLU}$
$RB_2 : [C_A(64, 5, 32), BN, \text{ReLU}, C_B(64, 5, 8), BN] + [C_S(64, 1, 8), BN], \text{ReLU}$
$RB_3 : [C_A(64, 5, 2), \text{ReLU}, C_B(64, 5, 1)] + [C_S(64, 1, 1)], \text{ReLU}$
$out(64, 1)$

D_s:	D_t:
$in_x(16,1)$, the conditional density $NI(64,1)\|in_y(64,1)$, the high-res input to classify	$2{*}in_y(64,3)$, the 3 high-res frames to classify
$C(32,4,32)$, leaky ReLU	$C(32,4,32)$, leaky ReLU
$C(16,4,64), BN$, leaky ReLU	$C(16,4,64), BN$, leaky ReLU
$C(8,4,128), BN$, leaky ReLU	$C(8,4,128), BN$, leaky ReLU
$C(8,4,256), BN$, leaky ReLU	$C(8,4,256), BN$, leaky ReLU
Fully connected, σ activation	Fully connected, σ activation
$out(1,1)$	$out(1,1)$

A.2 Parameters & Data Statistics

Below we summarize all parameters for training runs and data generation. Note that the model size includes compression, and we train the individual networks multiple times per iteration, as indicated below.

Details of generated results:

test	input size	tile (34^3) number	output size	time
Figure 3.14 a)	128^2	-	512^2	0.064s/frame
-	34^3	1	136^3	2.2s/frame
Figure 3.8 b)	64^3	8	256^3	17.9s/frame
Figure 3.9	$150 \times 100 \times 100$	96	$600 \times 400 \times 400$	234.48s/frame
Figure 3.10	$256 \times 180 \times 180$	441	$1024 \times 720 \times 720$	1046.07s/frame

Details of training runs for different models are listed in the following table. Our standard models that are used unless otherwise indicated are marked with a $^{(*)}$:

Train. iters	data: no. of sims, total frames	training and testing frames	low- high- input res ' res ' tiles	λ_{L_1}	$\lambda_f^{1,\dots,4}$
2D, Figure 3.14 a)$^{(*)}$	20, 4000	160, 40	$3*64^2, 256^2, 16^2$	3*5	-10^{-5} for all
2D, Figure 3.14 b)	20, 4000	160, 40			$10^{-4}/3, -10^{-4}/3, -10^{-4}/3, 10^{-4}/3$
2D, Figure 3.16 b)	20, 2400	320, 80			10^{-5} for all
3D, Figure 3.8 b)$^{(*)}$	20, 2400	96, 24	$2*64^3, 256^3, 16^3$	2*5	$-10^{-6}/3$ for all
3D, Figure 3.15 b)	20, 2400	96, 24			$10^{-6}/3, -10^{-6}/3, -10^{-6}/3, -10^{-6}/3$

Table Cont.	Trainings per step	Training Batch steps ' size	Model weights	Model size (Mb)	Training time (min)
2D, Figure 3.14 a)$^{(*)}$	2 for D_s, 3*2 for D_t, 2 for G	3*40k, 16	G, 634214 3*D_s, 706017 D_t, 706529	36.88	798.65
2D, Figure 3.14 b)				42.73	905.72
2D, Figure 3.16 b)				43.45	877.59
3D, Figure 3.8 b)$^{(*)}$	16 for D_s, 2*16 for D_t, 16 for G	2*7k, 1	G: 3148014 2*D_s: 2888161 D_t: 2890209	134.93	12636.22 2 GPUs
3D, Figure 3.15 b)				140.79	18231.97 2 GPUs

B Appendix of Chapter 4

Below we give details for the tests in Chapter 4.

Peak Test For the *Peak* test we generated a dataset of 110 images shown in Figure B.1. 55 images contain a peak located in the upper left corner of the image. The other 55 contain a peak located in the bottom right corner. We added random scribbles in the images to complicate the task. All 110 images were labeled with a one-hot encoding of the two possible positions of the peak. We use 100 images as the training dataset, and the remaining 10 for testing. All peak models are trained for 5000 epochs with a learning rate of 0.0001, with $\lambda = 1e-6$ for RR_{A}. To draw reliable conclusions, we show results for five repeated runs here. The neural network in this case contains one fully connected layer, with BN and ReLU activation. The results are shown in Figure B.2, with both peak modes being consistently embedded into the weight matrix of RR_{A}, while regular, autoencoder pretraining and orthogonal training show primarily random singular vectors.

We also use different network architectures in Figure B.3 to verify that the dominant features are successfully extracted when using more complex network structures. Even for two layers with BN and ReLU activations, our pretraining clearly extracts the two modes of the training data. The visual resemblance is slightly reduced in this case, as the network has the freedom to embed the features in both layers. Across all three cases, our pretraining clearly outperforms regular training and the orthogonality constraint in terms of extracting and embedding the dominant structures of the training dataset in the weight matrix It also yields lower LPIPS evaluations than autoencoder pretraining, which indicates features embedded in RR models represent the training data better.

MNIST Test We additionally verify that the column vectors of V_m of models from RR training contain the dominant features of the input with MNIST tests, which employ a single fully connected layer, i.e. $\mathbf{d}_2 = M_1\mathbf{d}_1$. In the first MNIST test, the training data consists only of 2 different images. All MNIST models are trained for 1000 epochs with a learning rate of 0.0001, and $\lambda = 1e-5$ for RR_{A}. After training, we compute the SVD for M_1. SVDs of the weight matrices of trained models can be seen in Figure B.4. The LPIPS scores show that features embedded in the weights of RR are consistently

Figure B.1: Dataset used for the *peak* tests

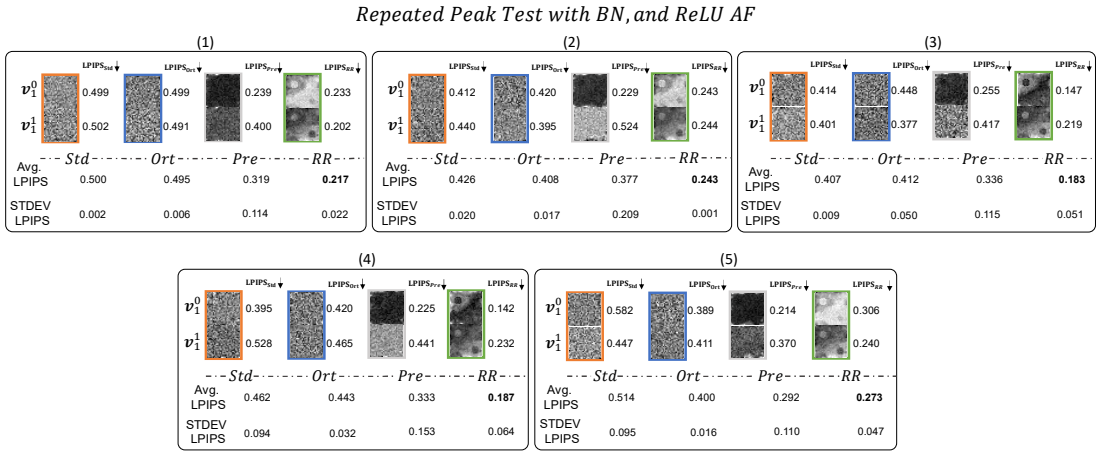


Figure B.2: Five repeated tests with the peak data shown in Sec. 3 of the main paper. RR_A robustly extracts dominant features from the dataset. The two singular vectors strongly resemble the two peak modes of the training data. This is confirmed by the LPIPS measurements

closer to the training dataset than all other methods, i.e., regular training Std, classic autoencoder pretraining Pre, and regularization via orthogonalization Ort. While the vectors of Std and Ort contain no recognizable structures.

Overall, our experiments confirm the motivation of our pretraining formulation. They additionally show that employing an SVD of the network weights after our pretraining yields a simple and convenient method to give humans intuition about the features learned by a network.

B.1 Mutual Information

This section gives details of the mutual information and disentangled representation tests from Sec. 4.2 of the main paper.

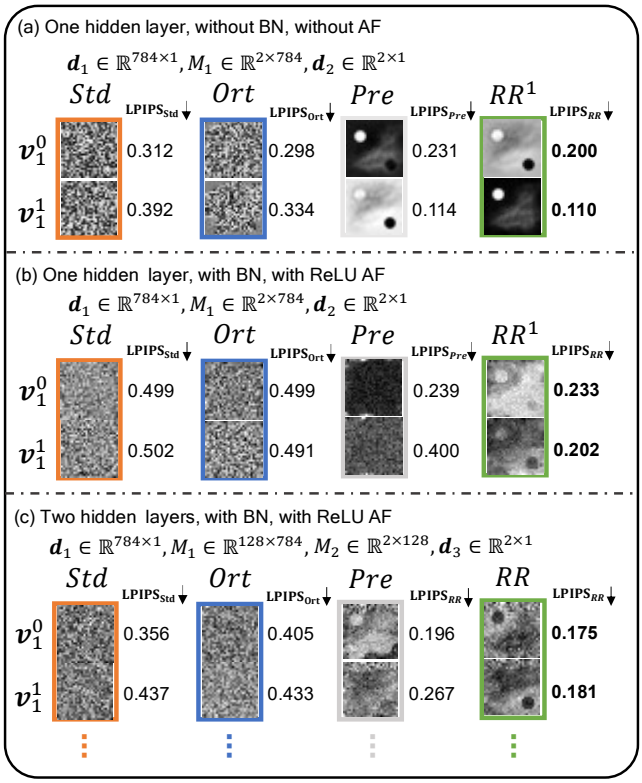


Figure B.3: Right singular vectors of M_1 for Across the three architectures, RR_A successfully extracts dominant and salient features

B.1.1 Mutual Information Test

Mutual information (MI) measures the dependence of two random variables, i.e., higher MI means that there is more shared information between two parameters. More formally, the mutual information $I(X;Y)$ of random variables X and Y measures how different the joint distribution of X and Y is w.r.t. the product of their marginal distributions, i.e., the Kullback-Leibler divergence $I(X;Y) = \text{KL}[P_{(X,Y)}||P_X P_Y]$, where KL denotes the Kullback-Leibler divergence. Let $I(X;\mathcal{D}_m)$ denote the mutual information between the activations of a layer $\mathcal{D}_m$ and input X. Similarly $I(\mathcal{D}_m;Y)$ denotes the MI between layer m and the output Y. We use *MI planes* in the main paper, which show $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$ in a 2D graph for the activations of each layer $\mathcal{D}_m$ of a network after training. This visualizes how much information about input and output distribution is retained at each layer, and how these relationships change within the network. For regular training, the information bottleneck principle [174] states that early layers contain more information about the input, i.e., show high values for $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$. Hence in the MI plane visualizations, these layers are often visible at the top-right corner. Later layers typically share a large amount of information with the output after training, i.e.

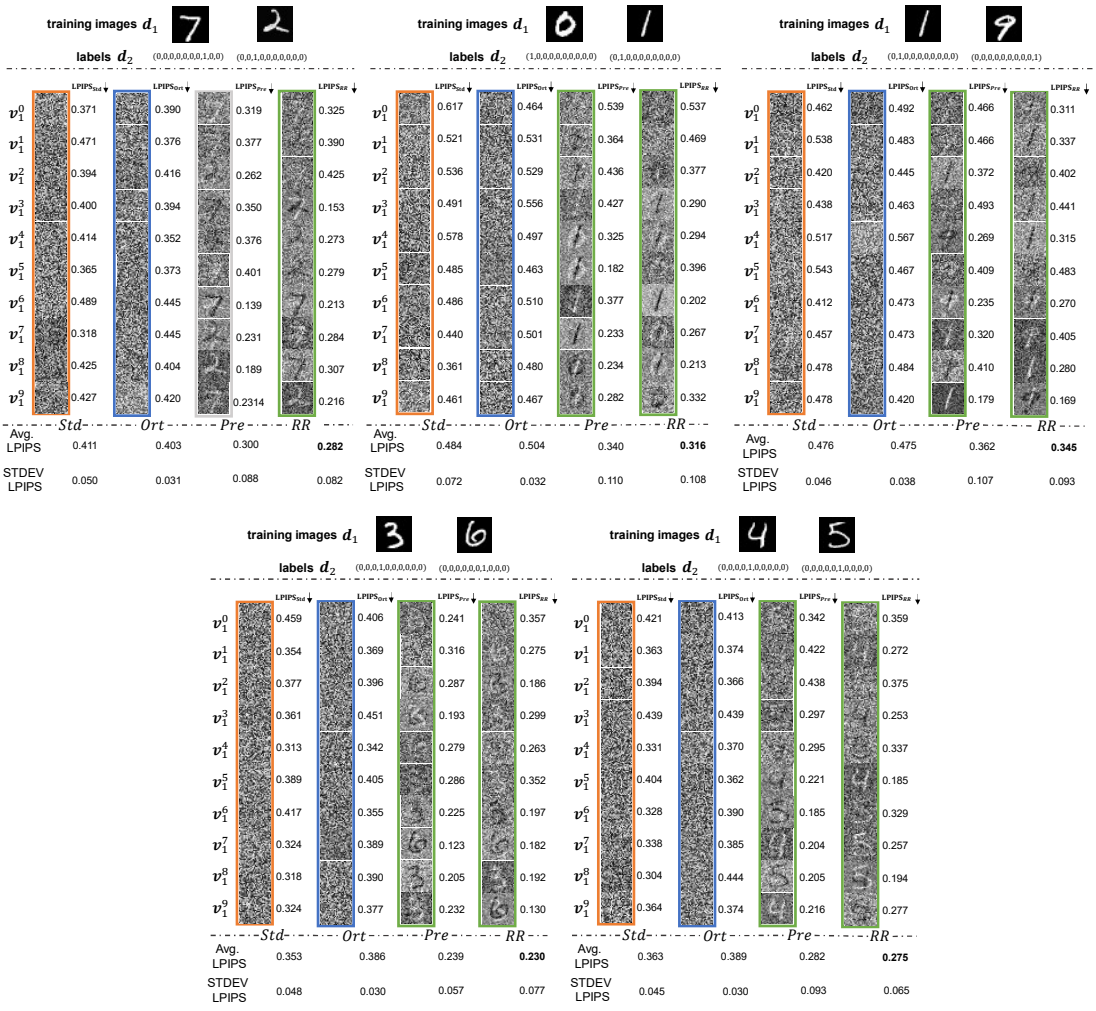


Figure B.4: SVD of the M_1 matrix for five tests with random two digit images as training data. LPIPS distances [147] of RR are consistently lower than Std and Ort

show large $I(\mathcal{D}_m; Y)$ values, and correlate less with the input (low $I(X; \mathcal{D}_m)$). Thus, they typically show up in the top-left corner of the MI plane graphs.

Training Details We use the same numerical studies as in [176] as task A, i.e. a regular feed-forward neural network with 6 fully-connected layers. The input variable X contains 12 binary digits that represent 12 uniformly distributed points on a 2D sphere. The learning objective is to discover binary decision rules which are invariant under $O(3)$ rotations of the sphere. X has 4096 different patterns, which are divided into 64 disjoint orbits of the rotation group, forming a minimal sufficient partition for spherically symmetric rules [208]. To generate the input-output distribution $P(X, Y)$, We apply the stochastic rule $p(y = 1 \mid x) = \Psi(f(x) - \theta), (x \in X, y \in Y)$, where Ψ is

a standard sigmoidal function $\Psi(u) = 1/(1 + exp(-\gamma u))$, following [176]. We then use a spherically symmetric real valued function of the pattern $f(x)$, evaluated through its spherical harmonics power spectrum [208], and compare with a threshold θ, which was selected to make $p(y = 1) = \sum_x p(y = 1 \mid x)p(x) \approx 0.5$, with uniform $p(x)$. γ is high enough to keep the mutual information $I(X;Y) \approx 0.99$ bits.

For the transfer learning task B, we reverse output labels to check whether the model learned specific or generalizing features. E.g., if the output is [0,1] in the original dataset, we swap the entries to [1,0]. 80% of the data (3277 data pairs) are used for training and rests (819 data pairs) are used for testing.

For the MI comparison in figure 4, we discuss models before and after fine-tuning separately, in order to illustrate the effects of regularization. We include a model with greedy layer-wise pretraining Pre, a regular model Std_A, one with orthogonality constraints Ort_A, and our regular model RR_A, all before fine-tuning. For the model RR_A all layers are constrained to be recovered in the backward pass. We additionally include the version RR^1_A, i.e. a model trained with only one loss term $\lambda_1 \left\| \mathbf{d}_1 - \mathbf{d}_1' \right\|_2^2$, which means that only the input is constrained to be recovered. Thus, RR^1_A represents a simplified version of our approach which receives no constraints that intermediate results of the forward and backward pass should match. For Ort_A, we used the Spectral Restricted Isometry Property (SRIP) regularization [166],

$$\mathcal{L}_{\text{SRIP}} = \beta\sigma(W^T W - I), \tag{B.1}$$

where W is the kernel, I denotes an identity matrix, and β represents the regularization coefficient. $\sigma(W) = sup_{z \in \mathbb{R}^n, z \neq 0} \frac{\|W_z\|}{\|z\|}$ denotes the spectral norm of W.

As explained in the main text, all layers of the first stage, i.e. from RR_A, RR^1_A, Ort_A , Pre_A and Std_A are reused for training the fine-tuned models without regularization, i.e. RR_AA, RR^1_AA, Ort_AA , Pre_AA and Std_AA. Likewise, all layers of the transfer task models RR_AB, RR^1_AB, Ort_AB , Pre_AB and Std_AB are initialized from the models of the first training stage.

Analysis of Results We first compare the version only constraining input and output reconstruction (RR^1_A) and the full loss version RR_A. figure 4 (b) of the main paper shows that all points of RR_A are located in a central region of the MI place, which means that all layers successfully encode information about the inputs as well as the outputs. This also indicates that every layer contains a similar amount of information about X and Y, and that the path from input to output is similar to the path from output to input. The points of RR^1_A, on the other hand, form a diagonal line. I.e., this network has different amounts of mutual information across its layers, and potentially a very different path for each direction. This difference in behavior is caused by the difference of the constraints in these two versions: RR^1_A is only constrained to be able to regenerate its input, while the full loss for RR_A ensures that the network learns features which are beneficial for both directions. This test highlights the importance of the constraints throughout the depth of a network in our formulation. In contrast, the $I(X;\mathcal{D})$ values of later layers

for Std_A and Ort_A exhibit small values (points near the left side), while $I(\mathcal{D};Y)$ is high throughout. This indicates that the outputs were successfully encoded and that increasing amounts of information about the inputs are discarded. Hence, more specific features about the given output data-set are learned by Std_A and Ort_A. This shows that both models are highly specialized for the given task, and potentially perform worse when applied to new tasks. Pre_A only focuses on decreasing the reconstruction loss, which results in high $I(X;\mathcal{D})$ values for early layers, and low $I(\mathcal{D};Y)$ values for later layers.

During the fine-tuning phase for task A (i.e. regularizers being disabled), all models focus on the output and maximize $I(\mathcal{D};Y)$. There are differences in the distributions of the points along the y-axis, i.e., how much MI with the output is retained, as shown in figure 4 (c) of the main paper. For model RR_AA, the $I(\mathcal{D};Y)$ value is higher than for Std_AA, Ort_AA, Pre_AA and RR^1_AA, which means outputs of RR_AA are more closely related to the outputs, i.e., the ground truth labels for task A. Thus, RR_AA outperforms the other variants for the original task.

In the fine-tuning phase for task B, Std_AB stands out with very low accuracy in figure 5 of the main paper. This model from a regular training run has large difficulties to adapt to the new task. Pre_A aims at extracting features from inputs and reconstructing them. Pre_AB outperforms Std_AB, which means features helpful for task B are extracted by Pre_A, however, it's hard to guide the feature extracting process. Model Ort_AB also performs worse than Std_B. RR_AB shows the best performance in this setting, demonstrating that our loss formulation yielded more generic features, improving the performance for related tasks such as the inverted outputs for B.

We also analyze the two variants of our pretraining: the local variant lRR_A and the full version RR_A in terms of mutual information. Figure B.5 shows the MI planes for these two models, also showing RR^1_A for comparison. Despite the local nature of lRR_A it manages to establish MI for the majority of the layers, as indicated by the cluster of layers in the center of the MI plane. Only the first layer moves towards the top right corner, and the second layer is affected slightly. I.e., these layers exhibit a stronger relationship with the distribution of the outputs. Despite this, the overall performance when fine-tuning or for the task transfer remains largely unaffected, e.g., the lRR_A still clearly outperforms RR^1_A. This confirms our choice to use the full pretraining when network connectivity permits, and employ the local version in all other cases.

B.1.2 Disentangled Representations

The InfoGAN approach [177] demonstrated the possibility to control the output of generative models via maximizing mutual information between outputs and structured latent variables. However, mutual information is very hard to estimate in practice [178]. The previous section and figure 4 (b) of the main paper demonstrated that models from our pretraining (both RR^1_A and RR_A) can increase the mutual information between network inputs and outputs. Intuitively, the pretraining explicitly constrains the model to recover an input given an output, which directly translates into an increase of mutual information between input and output distributions compared to regular training runs.

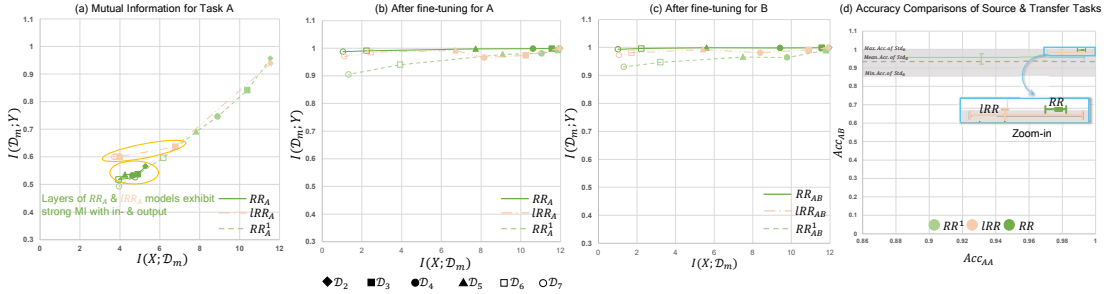


Figure B.5: MI plane comparisons for local ($\mathrm{lRR_A}$) versus full models ($\mathrm{RR_A}$). Points on each line correspond to layers of one type of model. **a)** MI Plane for task A. All points of $\mathrm{RR_A}$ and the majority of points for $\mathrm{lRR_A}$ (five out seven) are located in the center of the graph, i.e., successfully connect in- and ouput distributions. **b, c)** After fine-tuning for A/B. The last layer $\mathcal{D}_7$ of $\mathrm{RR_{AA}}$ builds the strongest relationship with Y. $I(\mathcal{D}_7;Y)$ of $\mathrm{lRR_A}$ is only slightly lower than $\mathrm{RR_{AA}}$. **d)** Accuracy comparisons among different models: $\mathrm{RR_{AA}}$ yields the highest performance, while $\mathrm{lRR_A}$ performs similarly with $\mathrm{RR_{AA}}$

For highlighting how our pretraining can yield disentangled representations (as discussed in the later paragraphs of Sec. 4 of the main text), we follow the experimental setup of InfoGAN [177]: the input dimension of our network is 74, containing 1 ten-dimensional category code c_1, 2 continuous latent codes $c_2, c_3 \sim \mathcal{U}(-1, 1)$ and 62 noise variables. Here, $\mathcal{U}$ denotes a uniform distribution.

Training Details As InfoGAN focuses on structuring latent variables and thus only increases the mutual information between latent variables and the output, we also focus the pretraining on the corresponding latent variables. I.e., the goal is to maximize their mutual information with the output of the generative model. Hence, we train a model $\mathrm{RR^1}$ for which only latent dimensions c_1, c_2, c_3 of the input layer are involved in the loss. We still employ a full reverse pass structure in the neural network architecture. c_1 is a ten-dimensional category code, which is used for controlling the output digit category, while c_2 and c_3 are continuous latent codes, to represent (previously unknown) key properties of the digits, such as orientation or thickness. Building relationship between c_1 and outputs is more difficult than for c_2 or c_3, since the 10 different digit outputs need to be encoded in a sinlge continuous variable c_1. Thus, for the corresponding loss term for c_1 we use a slightly larger λ factor (by 33%) than for c_2 and c_3. Details of our results are shown in Figure 4.6. Models are trained using a GAN loss [57] as the loss function for the outputs.

Analysis of Results In Figure 4.6 we show results for the disentangling test case. It is visible that our pretraining of the $\mathrm{RR^1}$ model yields distinct and meaningful latent space dimensions for $c_{1,2,3}$. While c_1 controls the digit, $c_{2,3}$ control the style and orientation of the digits. For comparison, a regular training run with model Std does result in meaningful or visible changes when adjusting the latent space dimensions. This illustrates

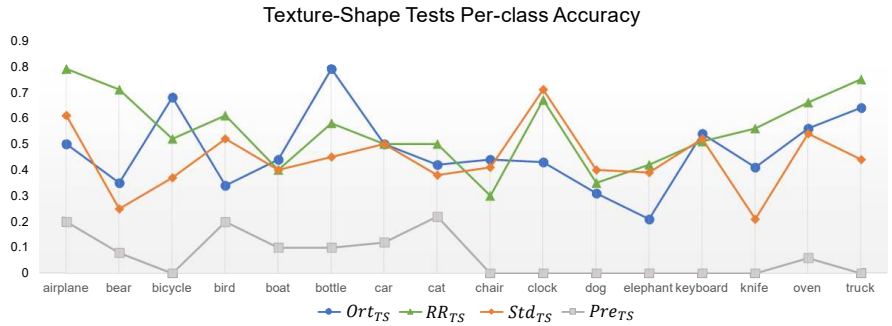


Figure B.6: Separate per-class test accuracies for the four model variants. The RR_{TS} model exhibits a consistently high accuracy across all 16 classes

how strongly the pretraining can shape the latent space, and in addition to an intuitive embedding of dominant features, yield a disentangled representation.

B.2 Details of Experimental Results

To ensure reproducibility, source code and data for all tests will be published. Runtimes were measured on a machine with Nvidia GeForce GTX 1080 Ti GPUs and an Intel Core i7-6850K CPU.

B.2.1 Texture-shape Benchmark

Training Details All training data of the texture-shape tests were obtained from [180]. The stylized data set contains 1280 images, 1120 images are used as training data, and 160 as test data. Both edge and filled data sets contain 160 images each, all of which are used for testing only. All three sets (stylized, edge, and filled) contain data for 16 different classes.

Analysis of Results For a detailed comparison, we list per-class accuracy of stylized data training runs for Ort_{TS}, Std_{TS}, Pre_{TS} and RR_{TS} in Figure B.6. RR_{TS} outperforms the other three models for most of the classes. RR_{TS} requires an additional 41.86% for training compared to Std_{TS}, but yields a 23.76% higher performance. (Training times for these models are given in the supplementary document.) All models saturated, i.e. training Std_{TS} or Ort_{TS} longer does not increase classification accuracy any further. We also investigated how much we can reduce model size when using our pretraining in comparison to the baselines. A reduced model only uses 67.94% of the parameters, while still outperforming Ort_{TS}.

B.2.2 Smoke Generation

Training Details The data set of the smoke simulation was generated with a Navier-Stokes solver from an open-source library [209]. We generated 20 randomized simulations

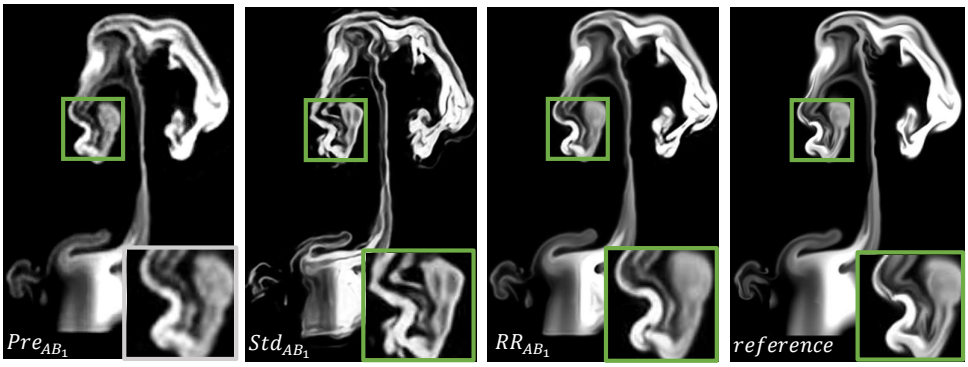


Figure B.7: Example outputs for Pre_{AB_1}, Std_{AB_1}, RR_{AB_1}. The reference is shown for comparison. RR_{AB_1} produces higher quality results than Std_{AB_1} and Pre_{AB_1}

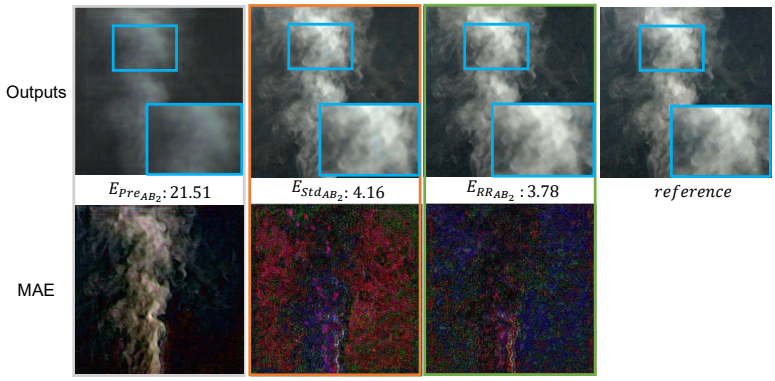


Figure B.8: Mean Absolute Error (MAE) comparisons for smoke task B_2 models. RR_{AB_2} shows the smallest error, and additionally achieves the best visual quality amongst the different models

with 120 frames each, with 10% of the data being used for training. The LR data were down-sampled from the HR data by a factor of 4. Data augmentation, such as flipping and rotation was used in addition. As outlined in the main text, we consider building an autoencoder model for the synthetic data as task B_1, and generating samples from a real-world smoke data set as task B_2. The smoke capture data set for B_2 contains 2500 smoke images from the ScalarFlow data set [183], and we again used 10% of these images as training data set.

Task A: We use a fully convolutional CNN-based architecture for generator and discriminator networks. Note that the inputs of the discriminator contain high resolution data $(64, 64, 1)$, as well as low resolution $(16, 16, 1)$, which is up-sampled to $(64, 64, 1)$ and concatenated with the high resolution data. In line with previous work [203], RR_{A} and $\mathrm{Std}_{\mathrm{A}}$ are trained with a non-saturating GAN loss, feature space loss and L^2 loss as base loss function. All generator layers are involved in the pretraining loss. As greedy layer-

wise autoencoder pretraining is not compatible with adversarial training, we pretrain Pre_A for reconstructing the high resolution data instead.

Task B_1: All encoder layers are initialized from RR_A and Std_A when training RR_{AB_1} and Std_{AB_1}. It is worth noting that the reverse pass of the generator is also constrained when training Pre_A and RR_A. So both encoder and decoder are initialized with parameters from Pre_A and RR_A when training Pre_{AB_1} and RR_{AB_1}, respectively. This is not possible for a regular network like Std_{AB_1}, as the weights obtained with a normal training run are not suitable to be transposed. Hence, the de-convolutions of Std_{AB_1} are initialized randomly.

Task B_2: As the data set for the task B_2 is substantially different and contains RBG images (instead of single channel gray-scale images), we choose the following setups for the RR_A, Pre_A and Std_A models: parameters from all six layers of Std_A and RR_A are reused for initializing decoder part of Std_{AB_2} and RR_{AB_2}, parameters from all six layers of Pre_A are reused for initializing the encoder part of Pre_{AB_2}. Specially, when initializing the last layer of Pre_{AB_2}, Std_{AB_2} and RR_{AB_2}, we copy and stack the parameters from the last layer of Pre_A, Std_A and RR_A, respectively, into three channels to match the dimensions of the outputs for task B_2. Here, the encoder part of RR_{AB_2} and the decoder of Pre_{AB_2} are not initialized with RR_A and Pre_A, due to the significant gap between training data sets of task B_1 and task B_2. Our experiments show that only initializing the decoder part of RR_{AB_2} (avg. loss:$1.56e7$, std. dev.:$3.81e5$) outperforms initializing both encoder and decoder (avg. loss:$1.82e7 \pm 2.07e6$), and only initializing the encoder part of Pre_{AB_2} (avg. loss:$4.41e7 \pm 6.36e6$) outperforms initializing both encoder and decoder (avg. loss:$9.42e7 \pm 6.11e7$). We believe the reason is that initializing both the encoder and decoder parts makes it more difficult to adjust the parameters for the new data set that is very different from the data set of the source task.

Analysis of Results Example outputs of Pre_{AB_1}, Std_{AB_1} and RR_{AB_1} are shown in Figure B.7. It is clearly visible that RR_{AB_1} gives the best performance among these models. We similarly illustrate the behavior of the transfer learning task B_2 for images of real-world fluids. This example likewise uses an autoencoder structure. Visual comparisons are provided in Figure B.8, where RR_{AB_2} generates results that are closer to the reference. Overall, these results demonstrate the benefits of our pretraining for GANs, and indicate its potential to obtain more generic features from synthetic data sets that can be used for tasks involving real-world data.

B.2.3 Weather Forecasting

Training Details The weather forecasting scenario discussed in the main text follows the methodology of the *WeatherBench* benchmark [210]. This benchmark contains 40 years of data from the ERA reanalysis project [145] which was re-sampled to a 5.625° resolution, yielding 32×64 grid points in ca. two-hour intervals. Data from the year of 1979 to 2015 (i.e., 324192 samples) are used for training. The benchmark also contains 165 years of historical simulation data from [185], and data from the year 1850 to 2005

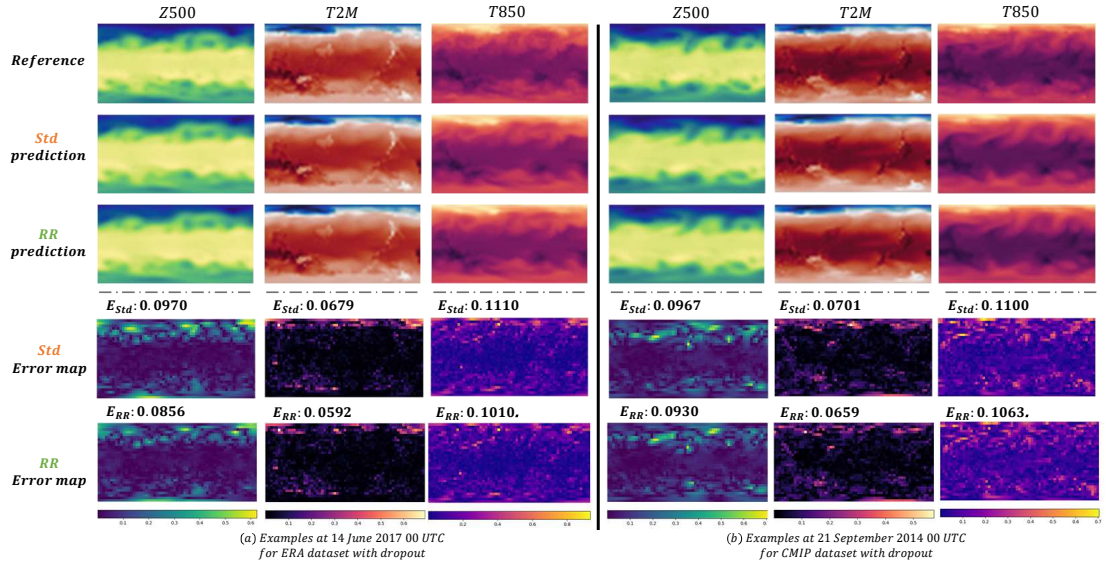


Figure B.9: MAE value comparisons between RR and Std(E_{RR} for RR and E_{Std} for Std). RR consistently yields lower errors than Std. The predictions inferred by the RR model are closer to the observed references

(i.e., 224672 samples) are used for training. All RMSE measurements are latitude-weighted to account for area distortions from the spherical projection.

The neural networks for the forecasting tasks employ a ResNet architecture with 19 layers, all of which contain 128 features with 3×3 kernels (apart from 7×7 in the first layer). All layers use batch normalization, leaky ReLU activation (tangent 0.3), and dropout with strength 0.1. As inputs, the model receives feature-wise concatenated data from the WeatherBench data for 3 consecutive time steps, i.e., t, $t - 6h$, and $t - 12h$, yielding 117 channels in total. The last convolution jointly generates all three output fields, i.e., pressure at 500 hPa (Z500), temperature at 850 hPa (T850), and the 2-meter temperature (T2M). Following [186], the learning rate was decreased by a factor of 5 when the loss did not decrease for two epochs, and the training is terminated after 5 epochs without improvements.

Analysis of Results In addition to the quantitative results given in the main text, Figure B.9 contains additional example visualizations from the test data set. A visualization of the spatial error distribution w.r.t. ground truth results is also shown. It becomes apparent that our pretraining achieves reduced errors across the whole range of samples. Both temperature targets contain a larger number of smaller-scale features than the pressure fields. The improvements of MAE from our pretraining approach are significant (c.a. 3% ˜10% across all cases), which represents a substantial improvement. The learning objective is highly non-trivial, and the improvements were achieved with the same limited set of training data. Being very easy to integrate into existing train-

ing pipelines, these results indicate that the proposed pretraining methodology has the potential to yield improved learning results for a wide range of problem settings.

C Appendix of Chapter 5

C.1 Training details

In this section, more details about temporal UV model training (section 5.2.4 of the main paper) will be illustrated. The DensePose model outputs UV coordinates with an extra I channel to classify different body parts. Below, we use I subscripts to denote the I channel of a UV coordinate, e.g., $G_I(P_t^r)$ refers to the I channel of $G(P_t^r)$. Then, for P_t^r from the DensePose model, we use an extra cross-entropy loss

$$L_I = -\boldsymbol{e}_{[P_t^f]_I} log G_I(P_t^r) \tag{C.1}$$

for I channel constraint, where $-\boldsymbol{e}_{[P_t^f]_I}$ is a one-hot vector indicating the $[P_t^f]_I^{\text{th}}$ I channel with $-\boldsymbol{e}_{[P_t^f]_{Ij}} = 1$ if $j = [P_t^f]_I$. We train G for DensePose P_t^r with the architecture shown in Figure C.1, and all of the discriminators D_s, D_t, and D_{img} follow the same encoder structure, as shown in Figure C.2. For UV data without an I channel, e.g., P_t^r generated from SMPL models, our pipeline is still applicable by training without L_i and removing the I channel part in G.

We apply gradient clipping for the gradients from L_s^{img} and L_t^{img} to stabilize the training of G. Parameters λ_2 and $\lambda_{uv,s}$ start from 200 and 10, respectively. They are decreased with rate 0.99 for every 1000 steps. On the other hand, $\lambda_{img,s}$, λ_{smo}, $\lambda_{uv,t}$, and $\lambda_{img,t}$ start from 0.001, 0.1, 1, and 1, respectively, but they are gradually increased with rate 1.01 for every 1000 steps.

C.2 Evaluation of results

In section 5.4 of the main paper, we follow [99] to evaluate temporal coherence of the results and estimate the differences of warped frames, i.e., T-diff $= \|I_{g_t}, \mathcal{W}(I_{g_t}, v_t)\|_1$. In our setting, we use the UV coordinates to calculate v_t, so that T-diff will purely be influenced by P_t. We first warp all the point coordinates $\mathbf{x}$ in I_{g_t} to the texture space, then we can calculate the displacement of all the points from I_{g_t} to $I_{g_{t+1}}$:

$$v_t^{texture} = \mathcal{W}(c_{img}, \omega_T(P_{t+1}^g)) - \mathcal{W}(c_{img}, \omega_T(P_t^g))), \tag{C.2}$$

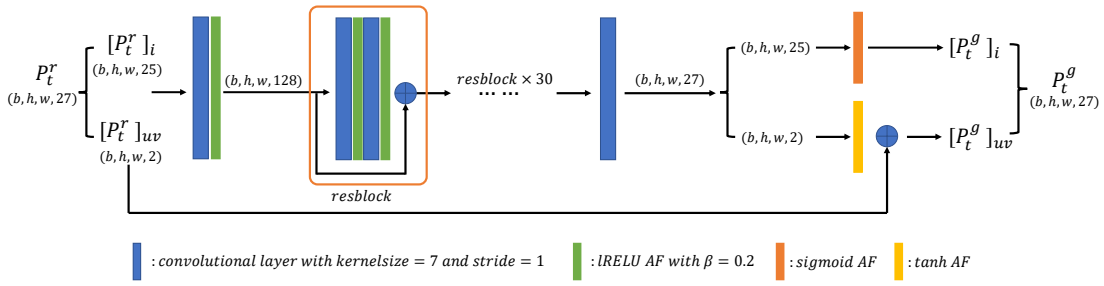


Figure C.1: Generator structure for training with P_t^r from the DensePose model. The corresponding part of I channel will be removed when training with P_t^r generated from the SMPL model.

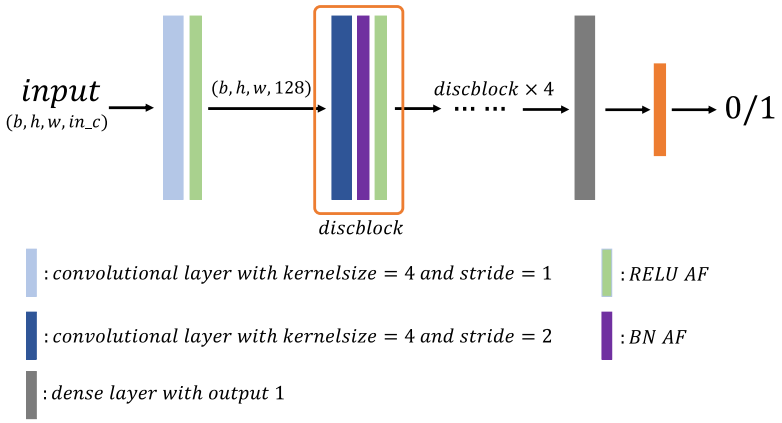


Figure C.2: Architecture of the discriminator networks, such as D_s, D_t, and D_{img}. Input channels in_c for D_s, D_t, and D_{img} are 2, 6, and 9, respectively.

	PSNR↑	LPIPS↓ $\times 10^{-2}$	tOF↓ $\times 10^{4}$	tLP↓ $\times 10^{-2}$	T-diff↓ $\times 10^{5}$
P_t^r	22.1	8.1	1.69	**1.0**	5.42
V_1	23.8	7.0	1.95	1.7	**4.33**
V_2	**23.9**	**6.7**	1.76	1.3	4.67
V_3	23.6	6.8	**1.68**	1.2	4.55

Table C.1: Quantitative comparisons between P_t^r and our different versions without cropping to fit $=P_t^r$. Our method show significant improvements for both spatial (PSNR and LPIPS) and temporal (tOF, T-diff) evaluation metrics. Evaluations with full shape lead to further improved PSNR and LPIPS evaluations for our results.

where $c_{img}(\mathbf{x}) = \mathbf{x}$. Then v_t can be obtained with:

$$v_t = \mathcal{W}(v_t^{texture}, \omega_I(P_{t+1}^g)). \tag{C.3}$$

In Table 5.1 of the main paper, we show quantitative comparisons between P_t^r and our different versions, which are made fair by cropping to fit $=P_t^r$. These results show improvements for both spatial and temporal evaluations. Here, we also show comparisons of those versions without cropping in Table C.1. We can see that our versions outperform P_t^r even further in terms of spatial quality. P_t^r performs the best with tLP, as P_t^r cannot generate the extended skirt and hair parts, which significantly decreases the area for evaluation. Here, we also can see that V_3 shows similar spatial quality as V_2 but an improved temporal coherence. Please refer to the supplementary video to see the improved temporal coherence of the synthesized sequence.