

Technical University of Munich, Germany
Chair of Computational Modeling and Simulation

Machine Learning - based Image Segmentation

Bernhard Mueller

Bachelor's thesis
for the Bachelor of Science program Civil Engineering

Author:	Bernhard Mueller
Supervision:	Prof. Dr.-Ing. André Borrmann
	Alexander Braun, M.Sc.
Issue Date:	April 2, 2018
Submission Date:	September 3, 2018

Abstract

Image processing tasks hold an important position in machine learning research today. Photogrammetric pictures of the physical environment contain high-level information about surrounding objects and, therefore, get processed in a variety of applications, like autonomous driving systems or medical examinations. The construction industry as well exhibits high potential of digitization, standardization, and automation of processes. This thesis proposes an approach for automated object detection and segmentation on construction site photos, based on a Convolutional Neural Network (CNN). The presented method allows a pixel accurate identification of objects like worksite elements on images of building sites. It offers an implementation of the Mask R-CNN (He et al., 2017) for the application in construction monitoring that allows almost real-time processing of photos or videos. The gained information can be used for the analysis of construction progresses and for quality assurance, for instance in the course of automated monitoring tools like progressTrack (Braun et al., 2018).

Based on a dataset of construction site photos captured by unmanned aircraft vehicles (UAVs), the network is trained to segment separate building elements, in the course of this thesis realized for formwork elements. The results contain a classification of the instances, a localization in the form of bounding boxes and a pixel by pixel segmentation of every detected object. It is implemented in Python, using a TensorFlow backend and the ResNet-101 (He et al., 2016) and FPN (Lin et al., 2017) backbone architecture of Mask R-CNN.

Zusammenfassung

Automatisierte Bildverarbeitung besitzt einen hohen Stellenwert in der Forschung des Maschinellen Lernens. Aufnahmen der Umgebung, die detaillierte Informationen über umliegende Objekte bereitstellen, werden bereits heute ausgewertet für eine Vielzahl von Anwendungen, wie Systemen des autonomen Fahrens oder medizinischen Untersuchungen. Auch die Bauindustrie besitzt ein hohes Potential der Digitalisierung, Standardisierung und Automatisierung von Prozessen. Diese Arbeit stellt eine Methode zur automatisierten Lokalisierung und Segmentierung von Objekten auf Baustellenfotografien vor. Auf Grundlage eines Convolutional Neural Networks (CNN) erfolgt die pixelgenaue Ausweisung von Objekten, wie beispielsweise Baubehelfselementen, auf visuellen Aufnahmen von Baustellen. Das entwickelte Programm wurde umgesetzt auf Grundlage einer auf das Bauwesen angewandten Implementierung des Mask R-CNN (He et al., 2017), das eine Verarbeitung von Bild- und Videomaterial nahezu in Echtzeit ermöglicht. Die Auswertung der Bildinformationen kann genutzt werden für Analysen des Bauprozesses sowie zur Qualitätssicherung, beispielsweise im Rahmen einer automatisierten Baufortschrittsüberwachung in Projekten wie progressTrack (Braun et al., 2018). Anhand eines Datensatzes von Baustellenfotos, bestehend aus Luftaufnahmen unbemannter Drohnen, wurde ein CNN trainiert auf die Segmentierung von Baustellenelementen, im Rahmen dieser Arbeit umgesetzt für Schalungselemente. Die Ergebnisse einer Bildverarbeitung beinhalten die seperate Klassifizierung aller erkannten Objekte, deren Lokalisierung sowie eine pixelgenaue Ausweisung auf dem entsprechenden Foto. Das CNN ist implementiert in Python, basierend auf einem TensorFlow Backend sowie der ResNet-101 (He et al., 2016) und FPN (Lin et al., 2017) Backbone-Architektur des Mask R-CNN.

Contents

List of Abbreviations

AI	Artificial Intelligence
AP	Average Precision
BIM	Building Information Modeling
CAD	Computer Aided Design
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DPM	Deformable Part Model
FAIR	Facebook AI Research
Fast R-CNN	Fast Region-based Convolutional Neural Network
FCIS	Fully Convolutional Instance-aware Semantic Segmentation
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LR	Learning Rate
ML	Machine Learning
mAP	mean Average Precision
MS COCO	Microsoft Common Objects in Context
MSE	Mean Squared Error
NN	Neural Network
R-CNN	Regions with CNN features
R-FCN	Region-based Fully Convolutional Network
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RoI	Region of Interest
RoIPool	Regions of Interest Pooling Layer
RPN	Region Proposal Network
SENet	Squeeze-and-Excitation Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VOC	Visual Object Classes

Chapter 1

Introduction

Buildings and construction sites are unique in geometry, structure and external conditions, however similar in planning and construction processes and workflows. Taking this fact into account, the construction sector exhibits high potential of process optimization by standardization and automation. Digital tools like Building Information Modeling (BIM) offer integrated approaches to unify planning operations, combine information and manage interfaces. The monitoring and documentation of construction progresses is a complex and time-consuming task as it commonly is realized manually by a construction manager. An automated and holistic documentation of the construction progress (Braun et al., 2015) simplifies these operations and can be used for further analyses: It enables comparisons of the as-built intermediate results to as-planned BIM models and documents states of the progress in temporal perspectives, thus allowing fully automated monitoring and defect identification at an early stage. Machine learning networks are powerful tools to automate differentiated tasks in a large number of application areas. Recently, Artificial Intelligence (AI) and Machine Learning (ML) are highly focused fields of research, whose development made much progress in the last few years. Computer vision as a branch of ML is applied in systems exemplary for autonomous driving, medical examinations or geodetical investigations. In civil engineering tasks as well, ML applications can contribute to the improvement of methods and processes.

This thesis proposes a ML approach for object detection and segmentation on construction site photos. It offers a Convolutional Neural Network (CNN) implementation of the Mask R-CNN (He et al., 2017), which allows the detection and segmentation of building elements during the construction workflow by means of captured construction site photos. The network is trained on a dataset of 747 aerial photographs of construction sites to detect construction objects, in the specific case of this thesis formwork elements. The construction elements get individually classified, localized and pixel-wise segmented in a nearly real-time process. The process- and computer vision-based software tool progressTrack (Braun et al., 2018) offers one approach for the standardization and automation of progress monitoring.

Braun et al. present a method in which photos of construction sites, captured by unmanned aerial vehicles (UAVs), are processed to reconstruct a 3D model of the construction progress. This 3D model is then compared to a BIM model of the project planning, to verify the matching of built and planned progress. In that context, the described network is applied to preprocess the captured photos and segment temporary components like formworks, which are then filtered out to improve the consistency of the processed 3D model.

The structure of the thesis is divided into two parts: The first part provides an overview of the theoretical background of ML. It explains the theory and mathematical concepts of supervised ML and the structure of NNs and CNNs. The following explains computer vision processes, including the development of CNN's for computer vision and state of the art networks of today. In specific, the task of instance segmentation and the Mask R-CNN network architecture are avowed, which is implemented in the approach of this thesis. The second step is a summary of the practical implementation. It describes the used dataset and illustrates the workflow of the implementation, finished by a discussion of the achieved results and an outlook toward the further procedure.

1.1 Related Work

Construction Monitoring Tools

Several studies of image processing on construction site photos have been conducted over the last few years. Wu et al. (Wu et al., 2010) offered an approach of object recognition based on 3D CAD and image filtering methods. Kim et al. (Kim et al., 2013) used image processing methods like mask filters or noise removal on photos for automated 4D CAD model updating. Chi and Caldas (Chi and Caldas, 2011) presented an estimate to detect mobile heavy equipment and workers on visual input data based on a NN, using videos from CCTVs on construction sites as a basis for automated safety monitoring systems. A method of finding concrete areas in photos, proposed by Zhu and Brilakis (Zhu and Brilakis, 2010) was done by use of parameter optimization. Implementations of Deep Neural Networks (DNN's) were presented by Gil et al. (Gil et al., 2018) who applied the Inception-v3 (Szegedy et al., 2015) for image classification by job-type, or Fang et al. (Fang et al., 2018) by a method detecting non-hardhat-use from surveillance videos based on an implementation of the Faster R-CNN (Ren et al., 2017). Kim et al. (Kim et al., 2018) used the R-FCN (Dai et al., 2016) architecture to detect construction equipment. Hamledari et al. (Hamledari et al., 2017) proposed a method of estimating the state of indoor building progress, which is done by a computer vision-based detection of components of under-construction indoor partitions. Cho et al. (Cho et al., 2018) presented an approach to assess real-time safety conditions of scaffolds by combining strain-gage sening with a SVM. Ha et al. (Ha et al., 2018) used a VGG-16 network (Simonyan and Zisserman, 2015) compared with a BIM to estimate an image-based indoor localization of users possessing a mobile device.

Image Segmentation

Image segmentation tasks can be categorized in the areas of semantic segmentation and instance segmentation. Whereas semantic segmentation CNN architectures process and segment the whole image as one instance, instance segmentation networks separate different image regions and process all detected objects individually (see 2.6.1). Semantic segmentation networks are offered in two methods: a processing and classification of separate pixels or a processing of the whole image at once. Pixel-wise implementations from Farabet et al. (Farabet et al., 2013) or Pinheiro and Collobert (Pinheiro and Collobert, 2014) used sliding windows on the images to classify and segment the pixels in small batches. As shared features were not reused between the overlapping patches, these models resulted to be ineffective. Long et al. (Long et al., 2015) proposed a method of processing whole images at once using a fully convolutional network. To reduce the number of parameters, the network performs downsampling and upsampling throughout the processing. In 2017, Lin et al. (Lin et al., 2017) presented the RefineNet, a multi-path refinement network for the improvements of tasks like semantic segmentation. By implementation of residual connections and residual pooling, RefineNet exploits information like high-level semantic features over the downsampling process. Today, the Mask R-CNN (He et al., 2017) as state of the art network architecture for instance segmentation outperforms other approaches made in this area. Among the further models for instance segmentations, selected CNN architectures are mentioned in the following: Badrinarayanan et al. (**?**) offered the SegNet architecture, an encoding-decoding framework with dilated convolutions. The SegNet used a novel technique of upsampling which involves storing the max pooling indices. Chen et al. (Chen et al., 2018) used ”atrous convolution”, convolution with unsampled filters to control the resolution and field of view of used filters. Ronneberger et al. (Ronneberger et al., 2015) offered the U-Net, a convolutional network for biomedical image segmentation. The FCIS method of Li et al. (Li et al., 2017) predicts a set of position-sensitive output channels simultaneously for object classes, boxes, and masks, making their network fast, thus exhibiting systematic errors on overlapping instances and creating spurious edges. Zagoruyko et al. (Zagoruyko et al., 2016) adapted the Fast R-CNN (Girshick, 2015) by skipping connections, creating a MultiPath network while methods from Bai et al. (Bai and Urtasun, 2017) or Arnab and Torr (Arnab and Torr, 2017) use semantic segmentation methods and cut pixels of the same category into different instances.

Chapter 2

Theoretical Background

2.1 Term of Machine Learning

Machine Learning (ML) constitutes a field of Artificial Intelligence (AI) and defines the study and modeling of learning processes and algorithms, that gives computers the ability to learn without being explicitly programmed. ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” (Michalski et al., 1983). The main idea of ML models is structured as follows: A model consisting of several algorithms is trained on input data, generating a decision structure to make as accurate as possible predictions applied on new data.

Early research in this area was done in the 1950’s, due to low computing power mostly of a theoretical nature. One of the first pioneers to be listed is Arthur Samuel, who worked on a successful machine learning approach for the game of checkers (Samuel, 1959). Since computing performance increased, research focused on more advanced models. Yann LeCun presented a noteworthy and pioneering research with the implementation of document recognition by means of gradient-based learning (LeCun et al., 1998) in 1998, which was a foundation stone for all following projects. In the last decade, machine learning became an increasing importance, as in the course of high computing power and modern technologies it presents a very powerful tool.

A glossary of common terms used in the illustrated areas Machine Learning, Convolutional Neural Networks and Computer Vision is documented in Appendix A.

2.2 Machine Learning Techniques

The first step of solving a problem using machine learning is to choose an appropriate model. Depending on the subjects being addressed, there are basically two different techniques of machine learning (MathWorks, 2016):

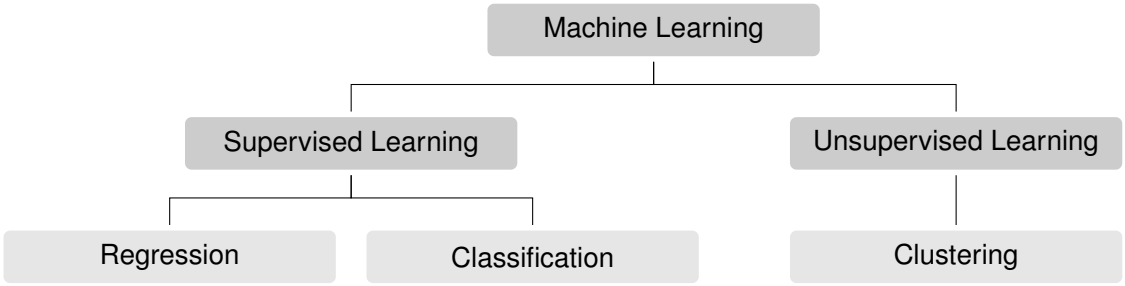

Figure 2.1: Machine learning techniques

Whereas supervised learning methods are used to generate reasonable predictions for the response to new data based on a known set of input data, unsupervised learning is applied to find hidden patterns or implicit structures in data (MathWorks, 2016).

2.3 Supervised Learning

A supervised learning model is a network that computes training data with given features and results for the requested output with the purpose to apply the learned relations on new data.

2.3.1 Model Structure

The basic model structure of a supervised learning model can be seen in figure 2.2:

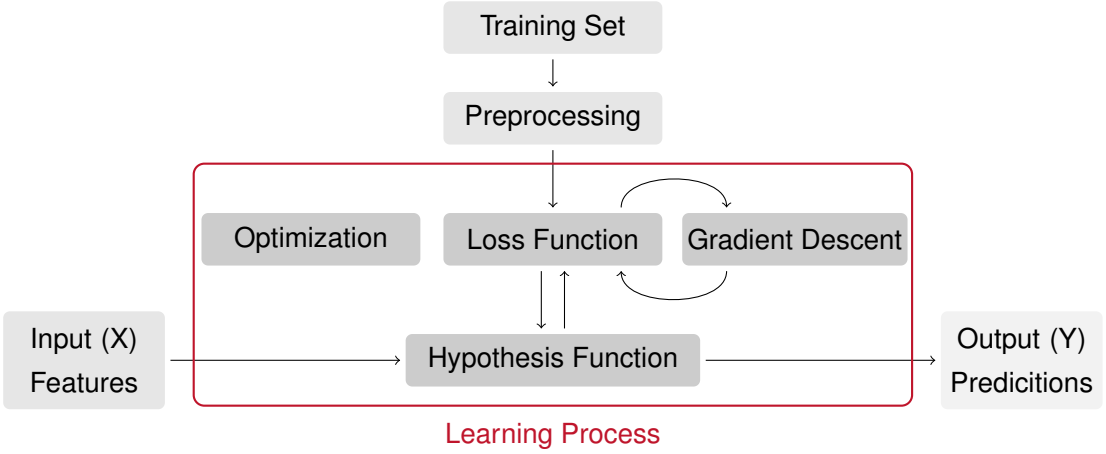

Figure 2.2: Supervised learning model structure

Hypothesis function

The objective of a supervised learning task is, given a training set, to learn a function $h : X \rightarrow Y$ so that $h(x)$ becomes a precise predictor for the corresponding value of Y, through use of the input features X. This function h is called a hypothesis function (Ng, 2012): Depending on the kind of problem, the given dataset, the input features and the model architecture, different approaches of hypothesis functions are used. A linear function $h(x)$ for n input features is mathematically defined as (Ng, 2012):

$$h(x) \quad = b + w_1 x_1 + ... + w_n x_n \quad = b + \sum_{i=1}^{n} w_i x_i \quad = \underbrace{b + w^T x}_{\text{vectorized notation}} \tag{2.1}$$

The features X are combined with different weights w, which get iteratively adjusted during the learning process to find the best working parametrization. The b term is called the bias term. The bias term is not multiplied with a feature and is implemented to initialize the hypothesis function. In addition to linear functions, several inclusions of the features are possible. An exemplary hypothesis function may be assembled by the mathematical terms:

$$h(x) = \quad b \; + \quad w_1 x_1 \; + \quad \underbrace{w_2 x_1^2}_{\text{polynomials}} \; + \quad \underbrace{w_3 x_1 x_2}_{\text{combinations}} \; + \quad \underbrace{w_4 \sqrt{x_1}}_{\text{roots}} \; + \quad \underbrace{w_5 \log x_1}_{\text{logarithms}} \tag{2.2}$$

Loss function

The loss function $L(w, b)$ defines a function that measures for each value of (w_i, b), how close the predicted outputs $h(x^{(i)})$ for every example $x^{(i)}$ from the training set are to the corresponding $y^{(i)}$ (Ng, 2012). Over the learning process of a model, the main objective is to optimize the hypotheses. The quality of the hypothesis function can be measured by the value of the loss function. Therefore, for the improvement of the hypothesis function, the loss function is minimized. A proven loss function which is commonly used in regression problems is for example the Mean Squared Error (MSE) function, computing the mean of the squared distances from $h(x)$ to Y (Ng, 2012):

$$L(w, b) = \frac{1}{m} \sum_{i=1}^{m} (h_{w,b}(x^{(i)}) - y^{(i)})^2 \tag{2.3}$$

Gradient descent

The loss function can be numerically minimized by an impact estimation of small variations of the hypothesis function parameter values on the loss function. This is measured by the gradient of the loss function with respect to the different weights $\theta_i = \{w_i, b\}$ (LeCun et al., 1998).

The simplest minimization procedure is the gradient descent algorithm, where the weights θ are iteratively adjusted as follows (Ng, 2012):

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} L(\theta) \tag{2.4}$$

All parameters θ_i get updated by the negative gradient of the loss function, derived after the according θ_i, which leads the loss function to a (local) minimum. The process of gradient descent iteratively repeats until the parameter updates converge to a threshold value near 0, in other words, until the loss function reaches the minimum. The parameter α is called the learning rate, one of the hyperparameters (Ng, 2012). The value of the learning rate determines the range of steps of adjustment in one iteration of gradient descent. In addition to gradient descent, more sophisticated methods can be used to minimize the loss function. A widely applied method is the Stochastic Gradient Descent (SGD) algorithm (Ng, 2012). In SGD, the parameter updates no longer occur after a computation of the loss function for all training examples $x^{(i)}$, but rather after the computation of the loss function for a single training example (Ng, 2012):

$$\text{for } i = 1 \text{ to } m \{ \qquad \theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} L^{(i)}(\theta) \qquad \} \tag{2.5}$$

In particular, at given training sets with a large number of training examples, stochastic gradient descent often is more computationally effective, even apart from the fact that one stochastic gradient descent update step, processing only one training example separately, not necessarily computes the steepest descent every time.

Optimization

During the learning process, a continuous adjustment of the model takes place, to map the factual research issue as good as possible. For the optimization of the network, there are different techniques and adjustment screws, the hyperparameters. The most important hyperparameters to be highlighted are the model architecture, the learning rate α and the regularization term.

Model architecture

The choice of a model architecture is the commencement of the development of a ML network. The first step is to build an initial architecture containing suitable functions to the problem. Whereas for example loss function algorithms like MSE are suitable for linear regression problems, approaches like Log Loss are implemented for classification problems (as further explained in 2.3.2 and 2.3.3). The properties of hypothesis function, loss function, and gradient descent algorithms have an important impact on the quality and efficiency of the network. As a second step during the learning process, modifications of the algorithms and comparisons to other functions get implemented, to evaluate the most efficient working combinations.

The functions and parameters to be the best in use derive not only from the required results and the computing capacity but also from the type of input data and the kind of features. The dataset has to be represented by input features that can be processed numerically. Data information like prices, volumes or weights are distinctively defined by their units and therefore can be implemented simply by value. The correct representation of information that is not clearly categorized requires higher effort. The feature interpretation for example of emotions in sentiment analysis is a sophisticated task, as these complex information are hardly comparable. In image detection problems, the data of the images are scanned and further processed by different filters (see 2.5.2). The evaluation of the most effective representation of features and the most important features being implemented also presents a task of optimization. In Neural Networks like CNNs, furthermore the number of neurons and layers have to be chosen and adjusted.

Learning rate

After choosing an architecture, in general the learning rate α (2.4) is the first hyperparameter to be adjusted. A learning rate that is too large and therefore adjusting too big steps in the weight updates can lead to a jump over the minimum of the loss function, whereas a learning rate being to small is computationally ineffective (Jacobs, 1988). A comparison of learning rate values for a gradient decent update of the loss as a function of the weights with one parameter w is exemplary shown in figure 2.3. Since the gradient descent algorithm numerically adjusts the weights, the size of the steps has great impact on its performance (Ng, 2012).

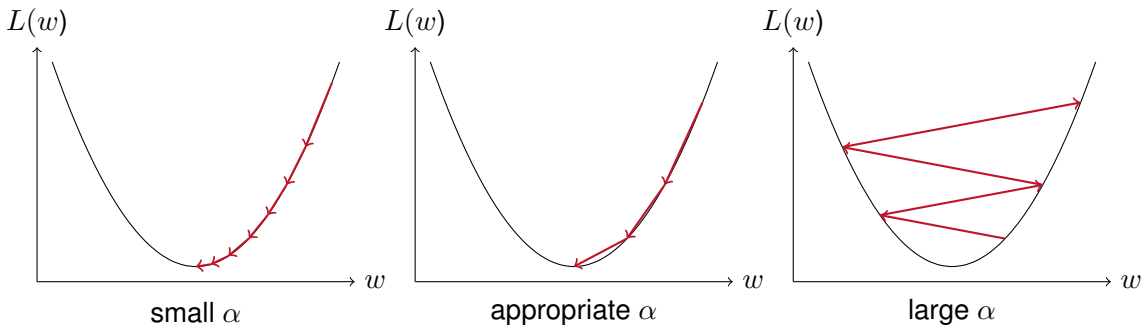

Figure 2.3: Comparison of learning rate values

Regularization

The hypothesis function $h(x)$ consists the bias term b and terms of the weights multiplied by features $w^T x$. For a good working hypothesis, an appropriate ratio of those two parts is highly important. Disproportional high influence of bias or rather the other weights leads to the phenomenom of an underfitting or overfitting hypothesis function (Ng, 2012).

Underfitting is referred to the situation when the bias term has too much impact on the hypothesis function. In figure 2.4 in the left plot, an example of a highly underfitting hypothesis function is illustrated. The plot shows output labels Y as a function of one input feature X for a training set containing 5 training examples. The hypothesis function underfitting the training set has a too powerful initialization term and, as a consequence, the parameters $w^T x$, that adapt the training examples, have nearly no impact.In other words, the function is underfitting the task, which frequently appears at a lack of features (Ng, 2012). A correction is possible by the implementation of new features, the combination of existing features (2.2) or by use of regularization.

The right plot of figure 2.4 shows an overfitting hypothesis function. Because of the high feature impact, the hypothesis function fits the training examples perfectly. By contrast applied on new data, the algorithm fails to make correct predictions. This phenomenon is based on the fact, that an overfitting hypothesis function does no longer represent the factional characteristic of the coherences in the data set (Ng, 2012). Overfitting often occurs at a lack of training data, whereas remedy can be provided by an extension of the data set, a reduction of the implemented features or by the use of regularization.

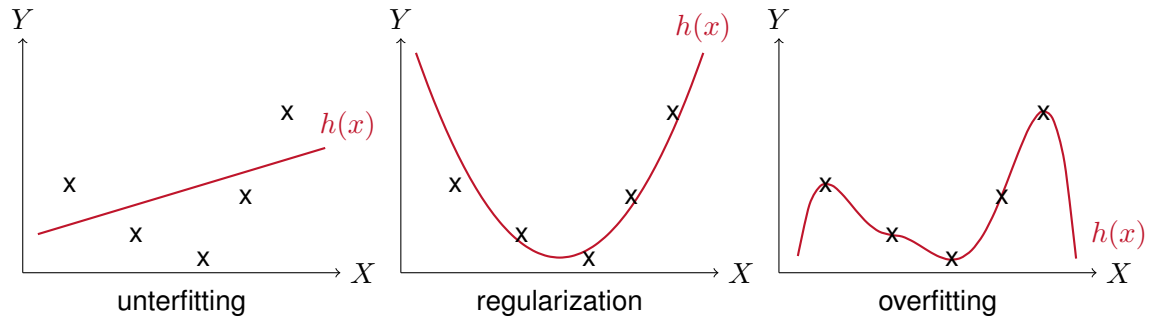

Figure 2.4: Underfitting vs. overfitting

Regularization methods are implemented tools to accomplish a balanced hypothesis function. A simple regularization technique is the implementation of a correction parameter λ, whose value regulates the proportion of impact of the single terms in the hypothesis function. The parameter λ can be implemented for example to the gradient descent algorithm (2.4) in addition to the loss function (Ng, 2018). For this application of regularization, the bias term b has to be separated to let the correction take effect:

$$b := b - \alpha \frac{\delta}{\delta w_i} L(w_i, b) \tag{2.6}$$

$$w_i := w_i - \alpha [\frac{\delta}{\delta w_i} L(w_i, b) + \frac{\lambda}{m} w_i] \tag{2.7}$$

Whereas a large value of λ reduces the impact of the weights w_i, a small value balances an oversized bias term.

2.3.2 Regression Problems

Supervised learning problems can be categorized into two different issues: regression problems and classification problems. The two tasks essentially differ in their type of model output. Classification models are built to classify data among different given categories. Therefore, the output of a classification model consists of a sum of discrete specified values. Regression models process functional values from continuous functions as an outcome. The possible number of output values of a regression model is thus not originated from a number of predefined values, but rather infinitely high. (Ng, 2012)
As an illustration, a classification task is for example to evaluate the likelihood of rain, possessing two output values: $[rain/no\ rain]$. A regression task would be to estimate the amount of precipitation, with arbitrarily output values of the unit $[mm]$.

Polynomial Regression

In regression problems, the main task is to compute a hypothesis function that adjusts the correlations between input features X and labels Y from training data. When new, unlabeled data are inserted, the function then calculates the likeliest predictions of the corresponding labels. A simple way to implement a regression problem is polynomial regression. In this linear model type consisting of one layer and therefore one hypothesis function and one loss function, polynomial hypothesis functions (2.2) and the MSE function (2.3) for computation of the loss are commonly used. A set of training examples and an appropriately learned hypothesis function may exemplarily appear as:

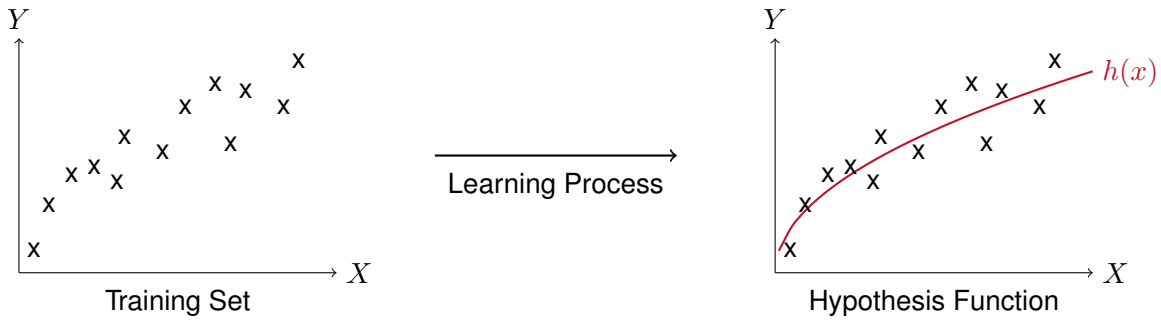

Figure 2.5: Regression function model

In nonlinear models like Convolutional Neural Networks (CNNs) with multiple layers, the principle of the regression model is the same: A hypothesis function is developed, which maps the problem statement as exactly as possible. In contrast to models with one layer, every neuron in the network contains a kind of hypothesis function, whose outputs get computed by the following neuron and so on. The processed results in this manner can be combined to create more complex connections.

2.3.3 Classification Problems

Classification problem models take over the classification and categorization tasks of data for training sets with known results. The classification model generates decision boundaries on the basis of labeled input data, constituted by a hypothesis function. Decision boundaries represent the policy for the classification of output estimations. A dataset with appropriately learned decision boundaries may exemplarily be from this shape:

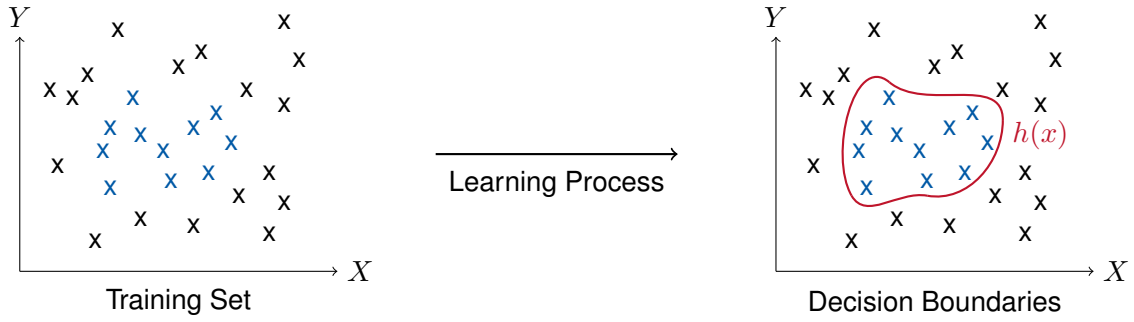

Figure 2.6: Classification function model

The output of a classification model is discrete-valued, whereby the number of output values is depending on the problem to be processed. There are two different classification tasks: binary classification and multi-class classification.

In a binary classification model, a binary classification is made. The classification consists of a positive and a negative class with an output of a single number in the range of $y \, \epsilon \, \{0, 1\}$. The value of this number expresses the probability of the positive class, a value of $y = 0.75$ as an example implies a probability of 75% for the predicted class. The computation of input features to such a value can be done by a logistic function like the sigmoid function (Ng, 2012), displayed in figure 2.7:

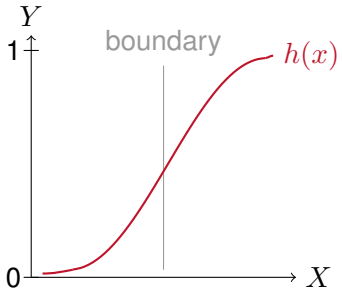

Figure 2.7: Sigmoid function

$$h(x) = \frac{1}{1 + e^{-w^T x}} \tag{2.8}$$

For every input feature, a probability is computed. The classification is then set up by a defined value of probability as a boundary. Whereas an input value far away from the boundary represents a high likelihood of one class, an input value close to the boundary provokes an uncertain prediction. Another approach of a classification hypothesis function is the Support Vector Machine (SVM) method, which is further explained in the next subsection.

Multi-class classifications categorize data in different classes of an arbitrary number. In the case of multi-class classification, two different possibilities of computation can be highlighted: A commonly used linear method of multi-class classification is the one vs. all principle. A one vs. all solution of a multi-class classification problem operates on the principle of one binary classification model for each class, which are computed consecutively. For each model, the two output values are firstly the class to be predicted and secondly all other classes in sum as a negative output. In a model with several hypothesis functions, like NNs, there are approaches with one classification model for all classes. In these approaches, an output prediction contains a vector with the length of the predicted classes, whose several items represent the probabilities for each class. Subsequently, the prediction for the classification is decided by the highest probability.

Support Vector Machines

Probably one of the best and commonly used classificaton hypothesis functions is the SVM learning algorithm (Ng, 2000). The main idea of SVMs is to generate a hyperplane classifier: The training data are mapped into a higher-dimensional feature space constructing a seperating hyperplane with maximum margins between the elements from the different classes, as shown in figure 2.8. This yields a nonlinear decision boundary in input space realized by a linear decision hyperplane (Schölkopf, 1998).

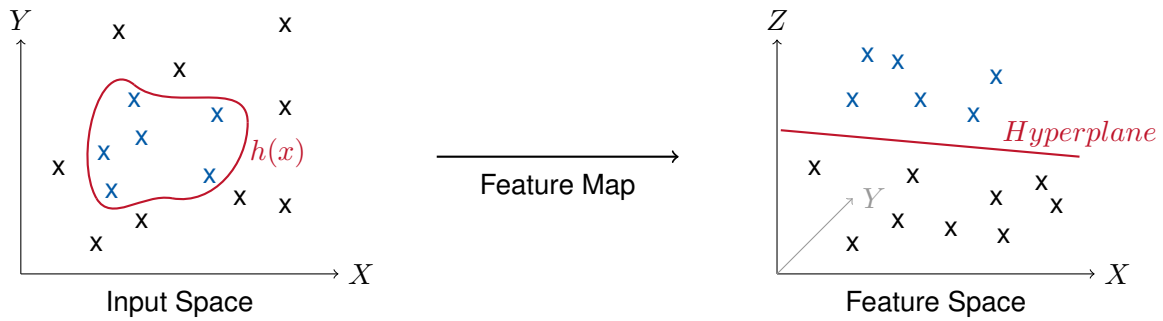

Figure 2.8: Feature space generation

For an output of two discrete values $y = \{-1, 1\}$ (positive or negative classified), the functional margin of a training example can be defined as (Ng, 2000):

$$\gamma^{(i)} = y^{(i)}(w^T x + b) \tag{2.9}$$

The prediction for the specific training example i is correct if $\gamma^{(i)}$ is positive. Considering the condition that y has the value (minus) one, the margin increases by a large $(w^T x + b)$ term, whereas a large margin means a high chance of correct prediction. The margin can be maximized by minimizing $||w||^2$ (Schölkopf, 1998).

The hypothesis function $h(x)$ is therefore extended by another dimension, a feature space computed by vector inner products. The combination of so-called kernels k, (nonlinear) approaches like radial basis functions or sigmoid functions instead of the linear function $(wx + b)$, corresponding with decision functions in the input space, the SVM hypothesis function can map an optimal margin classifier with a good performance on high-dimensional problems, containing numerous features (Schölkopf, 1998):

$$h_{w,b} = h(\sum_{i=1}^{n} k(x, x_{(i)}) + b) \tag{2.10}$$

2.4 Neural Networks

The combination of machine learning computations into a network is a commonly used technique in machine learning today. An artificial Neural Network (NN) is a computational model based on the structure of the brain (Kasneci et al., 1997). In a brain, billions of neurons are connected and work together according to the following principle: A neuron has many receptors, called dendrites, which receive input informations from other neurons. In the cell body, the informations get processed and exported by means of an output wire, the axon (figure 2.9). An artificial NN, like exemplary in figure 2.10, is an artificial approach of this structure, whereas an artificial neuron (figure 2.11) also operates in a simplified way like its natural archetype.

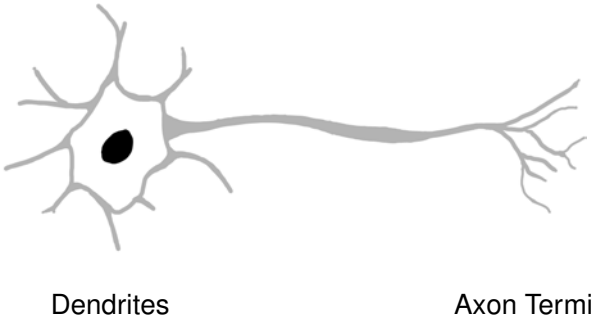

Figure 2.9: Neuron of a brain

Neural Networks from a technical view give a way of generating nonlinear, complex hypotheses. A neural network combines several ”neurons”, nodes of the network that all compute a kind of hypothesis function (Hagan et al., 1996). Using this technique, nonlinear patterns and decisions can be mapped. In figure 2.10, a possible architecture of a neural network is demonstrated.

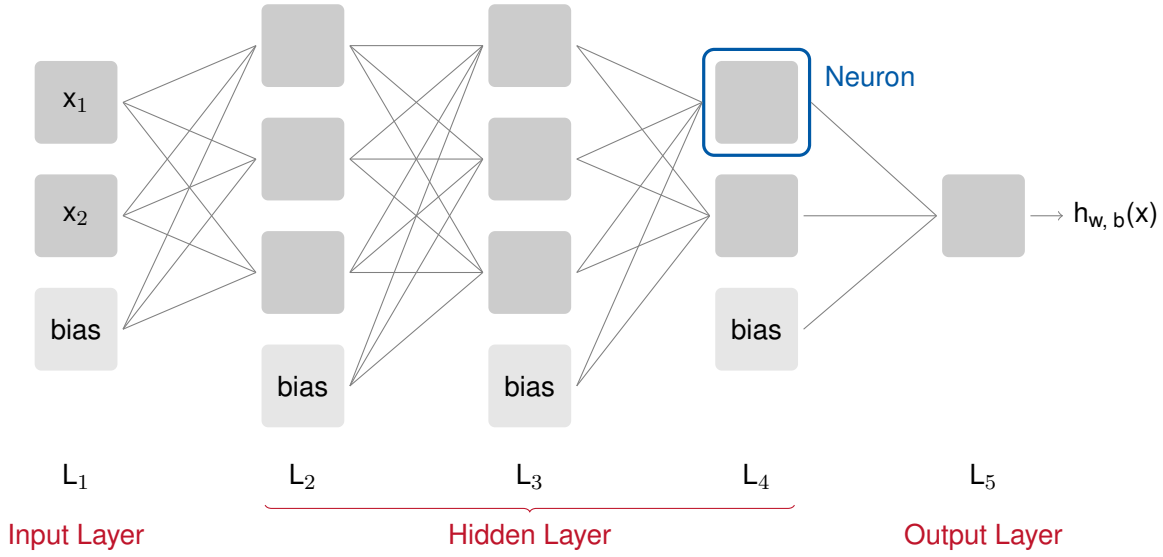

Figure 2.10: Neural network architecture

The input and output of a NN are similar to a linear model. In the first layer of the network, the input layer, all input features and a bias term are inserted. The following layers, in the architecture of figure 2.10 $L_{2\text{-}4}$, are called hidden layers. The term hidden layer refers to the characteristic that typically their input and output cannot be interpreted. In every neuron of the hidden layers, the computation of a hypothesis function, called activation function is done, which can be seen in figure 2.11 (Ng, 2017). The final layer, called output layer, computes interpretable values for the desired output. Depending on the task to be processed, the output layer computes an output of different sizes, for instance, a multi-class classification model (see 2.3.3) generates a vector with a length of the number of predicted classes. Thus, the output of a NN is a prediction for a requested y, computed in a high-grade nonlinear way. When all neurons in a layer are connected to all neurons in the previous layer, the layer is called a Fully Connected Layer.

2.4.1 Activation Function

Every neuron calculates a kind of hypothesis function, based on its input from the previous neurons. In general, two operations are executed: as a first step, the input values of the neuron x_i are combined with weights w_i, the neuron's parameters (see figure 2.11 (left)). As a second step, the adopted input z gets computed by an activation function $f(z)$, to be seen in figure 2.11 (right) (Ng, 2017). The activation function is a kind of classification hypothesis function that returns a positive value a for a positive z and an $a \to 0$ for a negative z. Frequently used activation functions are the sigmoid function (2.8), the tanh function or the Rectified Linear Unit (ReLU) function (Nair and Hinton, 2010)

$$f(z) = \max(0, z) \tag{2.11}$$

which is plotted in figure 2.12. The ReLU function outputs zero for every negative z and the value of z for every positive z. A neuron with implemented ReLU therefore gets activated by a positive result of z and does not get activated by computation of a negative z. In machine learning applications like Computer Vision tasks (2.6.1) using this technique, neurons that compare different patterns on an image get activated depending on the accordance to the respective pattern.

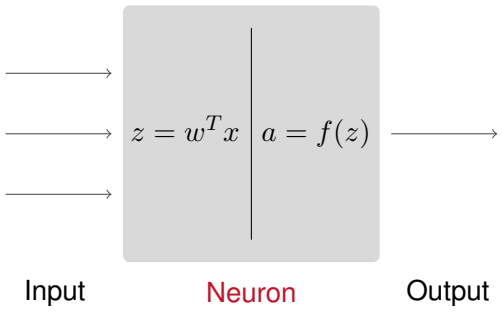

Figure 2.11: Computations of a single neuron

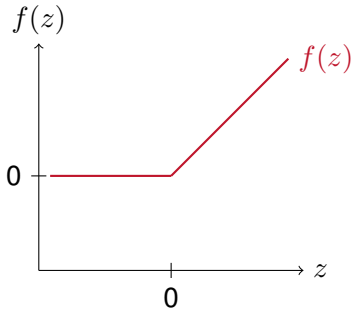

Figure 2.12: Rectified Linear Unit (ReLU)

2.4.2 Backpropagation

The training process of a NN is structured as follows: in a forward step, the input features get computed through all layers of the network, followed by a backward step optimizing the weights by use of gradient descent. At the end of the forward step, the network calculates a loss, on which all gradients have to be updated. Therefore, gradient descent is applied to all neurons, in other words on all weights of the NN. The gradients for the different layers can be computed in a backward pass. Starting at the last layer, where the loss is calculated by a loss function using the output $h_{w,b}(x)$ and requested y, the gradients of the different weights get calculated backward. This method is called backpropagation (Ng, 2017).
The weights $(W^{(i)}, b^{(i)})$ for any layer i consequently update in every iteration with use of a loss function L by the update rule (Ng, 2017):

$$W^{(i)} = W^{(i)} - \frac{\delta L}{\delta W^{(i)}} \tag{2.12}$$

$$b^{(i)} = b^{(i)} - \frac{\delta L}{\delta b^{(i)}} \tag{2.13}$$

In a training scenario, the network computes the forward and backward steps several times. In many epochs, the input features get processed, the results evaluated and the weights corresponding to the loss adopted until in best case, the loss converges to zero.

2.4.3 Weight Initialization

Before the training process starts, the weights of the network have to be initialized. A lack of initialization and therefore all weights set to zero causes problems in the backpropagation step as all gradients will be zero (Ng, 2017). An initialization with unique values of the weights also is not expedient: After computing the input features in the first layer, the activations of all neurons would be the same as the calculation was done with same values for their weights. This effect will occur at all layers of the NN, a phenomenon called symmetry (Ng, 2017). To avoid symmetry in the network, all weights get initialized with different values (near zero). A simple approach would be a random initialization. In Deep Neural Networks (DNNs), networks with a large number of layers, it is important to scale the values of initialization, depending on the depth of the layers (Glorot and Bengio, 2010): a forward calculation through the layers has a multiplicative effect on the weights, as small values processed over many layers converge toward zero. To maintain activation and gradient variances, a normalized initialization is done. A commonly used technique is Xavier initialization (Ng, 2017):

$$w^{(i)} \sim N(0, \sqrt{\frac{2}{n^{(i)} + n^{(i+1)}}}) \tag{2.14}$$

where n represents the number of neurons.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are specifically designed to process the variability of multidimensional shaped input (LeCun et al., 1998) as a CNN is a form of a NN with convolutionally extracted features. A CNN, also known as ConvNet, basically consists of three different layer types: Convolutional (Conv) layers (2.5.2), Pooling layers (2.5.3) and Fully connected (FC) layers (2.5.4). A possible model architecture of a CNN is exemplarily illustrated in figure 2.13.

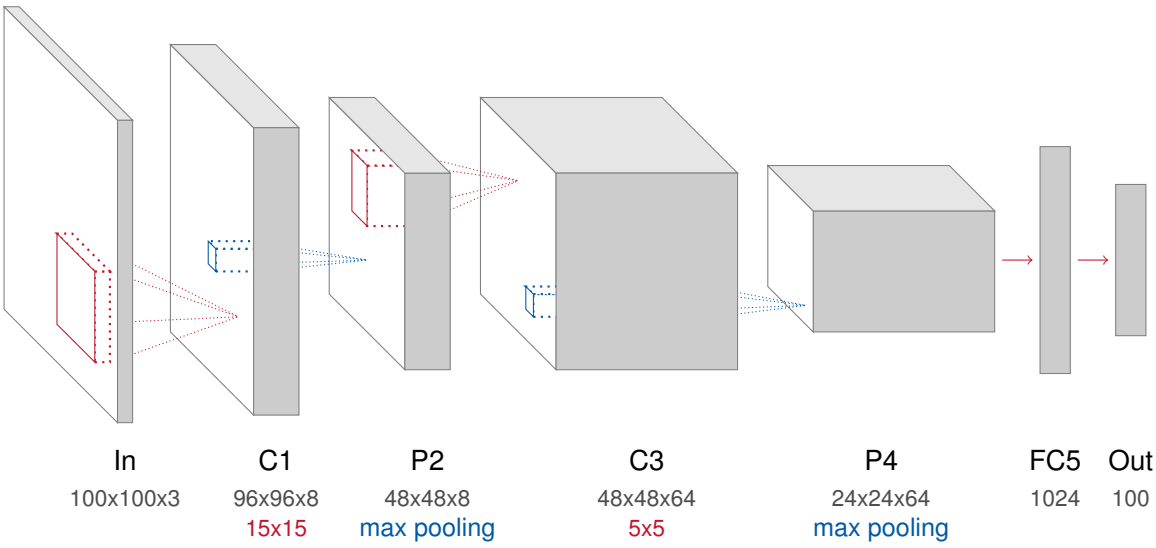

Figure 2.13: Convolutional neural network architecture

2.5.1 Feature extraction

A NN inputs all features as single values x_i (see figure 2.10). The input is therefore a vector by dimension [nx1] with n as the number of input features. In tasks like image processing, the input of a network typically is an image, whose pixels are the processed features. To be able to process images in a NN, the features would have to be extracted to a vector by stringing together all the pixels. This procedure does not scale larger images, as for example an image of size [100x100x3] (100 pixels width, 100 pixels height, 3 color channels (RGB)), see figure 2.13 "In", would lead to neurons with 100*100*3 = 30.000 weights each. For larger images, this fully connected structure due to the increasing number of weights would be computationally expensive and can lead to overfitting problems if training data are scarce (LeCun and Bengio, 1995).

The approach of CNNs is to process input features no longer unfolded, as rather in their original shape (Ng, 2017): The input data, for instance, the image of figure 2.13, is processed as a matrix by dimension [100x100x3]. The weights to compute the input features are also no longer a vector, but rather a matrix that is combined with the matrix of input features of the input image. Instead of computing the features in a fully connected layer architecture, the idea is to build a locally connected network: The neurons are locally connected to a small region of the previous layer, in the case of image processing, to a region of pixels.

The weight matrices have a smaller dimension than the input matrix and ”scan” the image as so-called filters (2.5.2). As in every Conv layer the neurons are combined with a defined region of the previous layer, the deeper a hidden layer in the CNN is located, the larger its connected area to the initial input gets. Feature extraction takes place over all Conv layers and Pool layers usually up to an FC layer, that fully combines all information and calculates final predictions. Theoretically, an arbitrary number of Conv layers and Pool layers can be combined in a CNN, finally processed by an FC layer as output layer. A CNN containing a huge number of hidden layers is called a Deep Neural Network (DNN).

2.5.2 Convolutional Layer

A Convolutional (Conv) layer computes input from previous layers, partitioned in local regions by the separate neurons (Ng, 2017): A matrix filled with weights slides over all features of the previous layer and computes a value in every step of moving. The size of the matrix specifies the receptive field of each neuron (Karpathy et al., 2016): Every neuron is locally connected to an input volume of the previous layer by the size of the matrix. Applied matrices can vary in width and height, but always have the same depth as the input. The matrix size and therefore the receptive field can be varied during the learning process and is one of the hyperparameters of the network. Another hyperparameter is the depth of the Conv layer: it corresponds to the number of scanning matrices (filters) that are applied. Every matrix produces an output, often called activation map (see figure 2.15). In the network of figure 2.13, the Conv layer C1 contains 8 filters by dimension [15x15x3] that produce 8 activation maps and therefore possesses a depth of 8, whereas C3 is assembled by 64 activation maps.
The matrix in figure 2.15 by dimension [2x2x1] contains $2 * 2 * 1 = 4$ weights. As the output contains 4 neurons, the activation map would consist of $4 * 4 = 16$ weights + 1 bias. Processing larger input volumes and using scanning matrices with a larger dimension leads to a huge number of weights in every layer. As an example, the Conv layer C1 of figure 2.13 using different weights for every neuron would consist of

$$98 \textit{ (width C1)} * 98 \textit{ (height C1)} * 15 * 15 * 3 \textit{ (filter dimension)} * 8 \textit{ (number of filters)} = 51.861.600$$

weights plus additional bias. To reduce calculation effort, CNNs use parameter sharing (Karpathy et al., 2016). Instead of setting different weights for every neuron of the activation map, one matrix, called filter, uses the same weights on all neurons. The number of weights thus decreases to

$$15 * 15 * 3 \textit{ (filter dimension)} * 8 \textit{ (number of filters)} = 5.400$$

and, with adding a bias for every filter, to 5.408 parameters.

Parameter sharing is in specific a reasonable method when every point of an image shall be scanned for the same patterns and thus by identical weights for every neuron in one activation map. An activation map in the Conv layer can then be computed as a convolution of the neuron's weights with the input volumes which is, furthermore, eponymous for the layer.
The application of different weight sets as a filter (also called kernel) is a commonly used method. Figure 2.14 shows example filters learned by Krizhevsky et al. in the AlexNet CNN architecture (Krizhevsky et al., 2012). Each of the 96 filters is by size [11x11x3], shared by 55*55 neurons in every activation map and learned by the first Conv layer on [224x224x3] images.

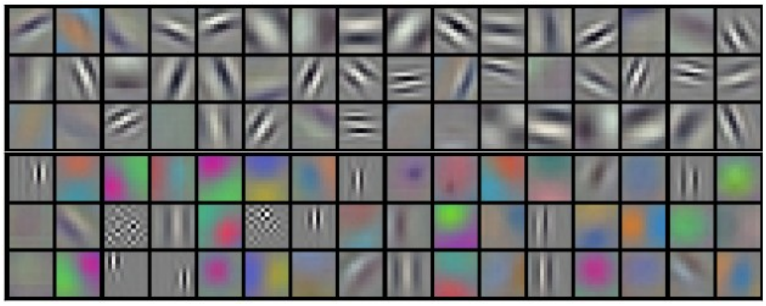

Figure 2.14: Filter examples, from (Krizhevsky et al., 2012)

An example of the computation steps in a Conv layer with one filter by dimension [2x2x1] processed on a [3x3x1] input image can be seen in figure 2.15:

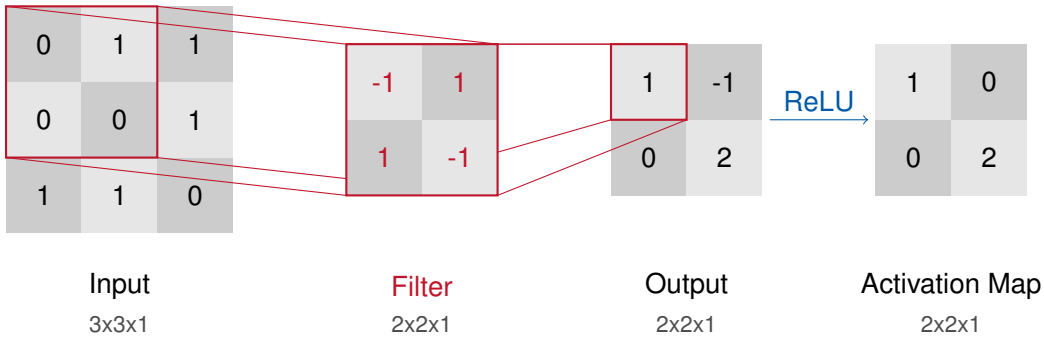

Figure 2.15: Calculation steps in an activation map

The receptive field of the neurons is set to [2x2x1], so in every computation step, a [2x2x1] region of the image gets processed. The calculation is done by an elementwise multiplication of the input values with the filter values, which is subsequently summed up to produce a single output value (Karpathy et al., 2016). One exemplary computation is highlighted in red in figure 2.15. The calculation is implemented in CNNs by means of a matrix multiplication. The output matrix then usually gets processed by an activation function like the tanh function, the sigmoid function (2.8) or as in figure 2.15 by use of the ReLU function (2.11). Scanning the images, a filter like shown in figure 2.14 gets activated, in other words, calculates high values in its activation map when the pattern in the image is similar to the filter.

Spatial Arrangement

Three hyperparameters determine the dimension of a Conv layer (Karpathy et al., 2016): the depth, the stride and the value of zero-padding. Usually, as deeper located layers in the CNNs decrease in width and height, they increase in depth to balance the amount of information. The depth of a Conv layer depends on the number of filters of the layer. The width and height hinge on the stride, with which a filter slides over the input and the value of so-called zero-padding. With a stride of 1, the filter moves one pixel after every computation. A stride of 2 leads the filter to move two pixels at a time. An increasing number of the stride produces smaller output volumes spatially and reduces the overlap of processed image regions. Figure 2.16 shows a [2x2x1] filter output with stride 1 (top) compared with a stride of 2 (bottom), applied on an [4x4x1] input image.

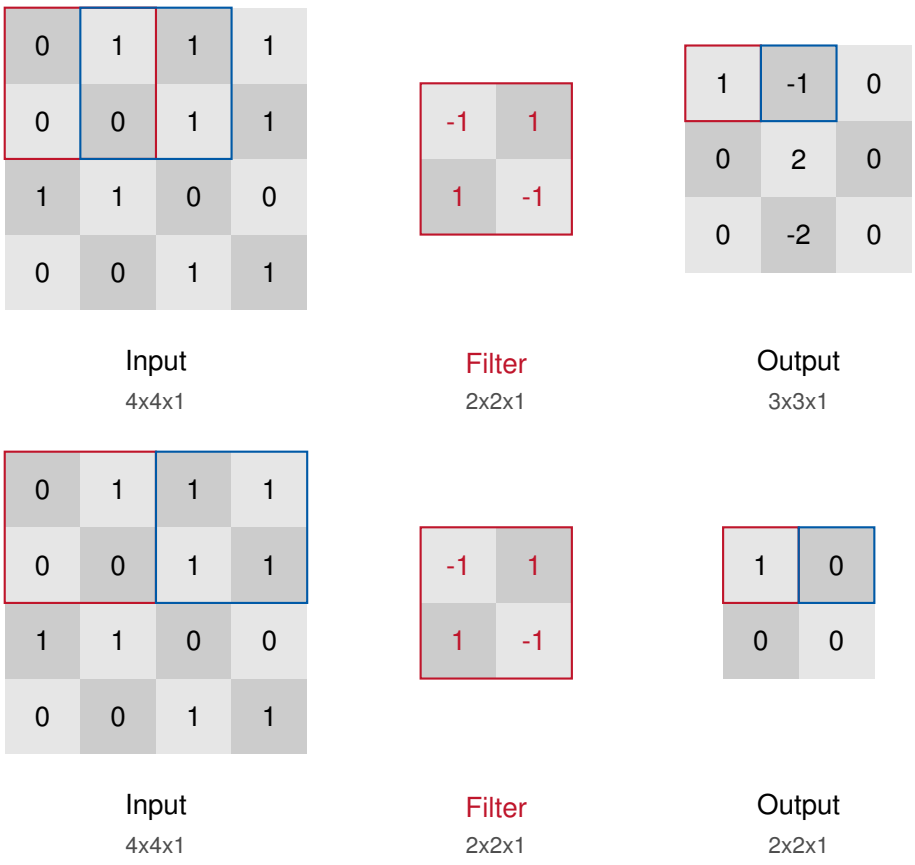

Figure 2.16: Filter calculation: stride 1 (top) vs. stride 2 (bottom)

The input size must be divisible by the filter stride, as only a natural number of steps can be executed (Karpathy et al., 2016). A technique to adjust the input size is zero-padding: Along the borders of the input, rows with values set to zero are attached to pad the input to a bigger size. Noticing, a (commonly used) pixel overlapping stride higher weights all pixels as they got

processed multiple times in various steps, except the pixels at the borders, zero-padding also balances this effect. Figure 2.17 shows an example with zero-padding of an amount of 1 applied to a [2x2x1] filter, computing a [2x2x1] image with a stride of 2, achieving a same-dimensional output.

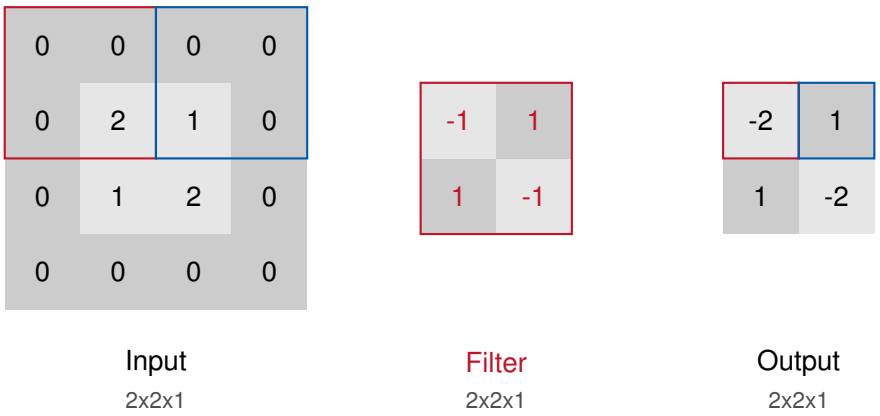

Figure 2.17: Zero-padding

The spatial size of an activation map can be computed as a function $n(W, F, P, S)$ of the hyper-parameters input volume size W, receptive field of the neurons F, amount of zero-padding P and the value of the stride S (Karpathy et al., 2016):

$$n(W, F, P, S) = \frac{W - F + 2P}{S} + 1 \tag{2.15}$$

2.5.3 Pooling Layer

As a CNN contains a huge number of parameters, which increases massively by increasing the layer dimensions, it is important to reduce the number of weights and neurons to better the computational performance and prevent overfitting problems. A commonly used method to progressively reduce the spatial size of the representation are Pooling layers, that are inserted in-between successive Conv layers in a CNN. A pooling layer operates independently on every activation map of the input and resizes it spatially (Karpathy et al., 2016). In the CNN architecture showed in figure 2.13, two Pooling layers P2 and P4 are implemented, using the technique of max pooling. A Pooling layer does not contain any parameters but reduces the input volume by means of mathematical operations. The size of an input volume by dimension [W_1xH_1xD_1], decreased by a Pooling layer, can be computed, with given receptive field of the neurons (pooling filter size) F and the stride S, by the formula (Karpathy et al., 2016)

$$W_2 = \frac{W_1 - F}{S} + 1 \qquad H_2 = \frac{H_1 - F}{S} + 1 \qquad D_2 = D_1 \tag{2.16}$$

whereby the height and width of the volume decreases and the depth remains unchanged.

Commonly used methods for pooling are max pooling (Zeiler and Fergus, 2014) or average pooling (Boureau et al., 2010). In a max pooling operation, at every step, the filter takes the maximum value over the processed input values, discarding all other activations. The most common form of pooling is a max pooling layer with filters by size [2x2] and a stride of 2 (Karpathy et al., 2016). In average pooling, the filter computes the average values of the processed input values. Figure 2.18 shows a max pooling operation compared to an average pooling operation, for a filter by size [2x2] applied on an input volume by size [4x4].

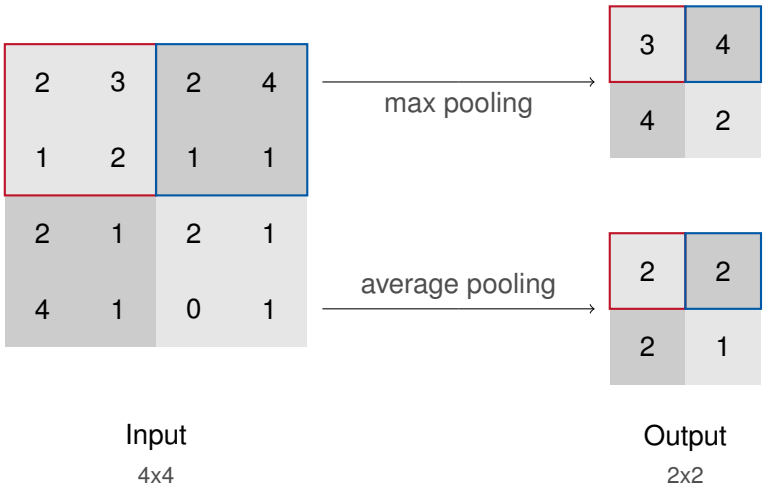

Figure 2.18: max pooling vs. average pooling

2.5.4 Fully Connected Layer

The last layer of a CNN commonly is a Fully Connected (FC) layer. As with ordinary NNs, in a Fully Connected layer, every neuron is connected to all elements of the previous layer. An FC layer at the end of a CNN is used to compute the final predictions and, therefore, is by dimension of the desired output, for example for multi-class classifications it has the shape of a vector. In the architecture of figure 2.13, FC5 by size [1024x1x1] is implemented to predict values for a multi-class classification consisting of 100 classes. The values of the FC layer finally are computed by an interpreting function, in the case of classification for instance by a softmax function or a SVM.

Optimization

As it is computationally expensive to compute the initial gradient descent in every backpropagation step, it is common in CNNs to use smaller batches for gradient learning, as the SGD algorithm (2.2). An approach of SGD that usually produces higher converge rates in deeper networks is called Momentum update (Sutskever et al., 2013), a technique that accumulates a velocity vector in direction of the update steps across iterations and combines this vector with the new update steps. Momentum update steps ”smooth out” backpropagation. A commonly used method for regularization for CNNs, in addition, is dropout (Srivastava et al., 2014), that randomly suspends neurons in every training step.

2.6 Computer Vision

2.6.1 Tasks

Computer vision can be described as the computational transformation of data from images or videos into either a decision or a new representation. Whereas the human brain divides visual signals into different channels with separate kind of information, a computer vision system computes a grid of input values that have to be interpreted, patterned and transformed by different numerical operations (Bradski et al., 2013). As it is nearly impossible to hard code such operations, computer vision is a highly focused research area in machine learning. Currently, huge improvements for many different computer vision tasks are obtained, which can exemplarily be seen in selected projects of 1.1.
The main fields of activity in computer vision can be categorized into three core areas: classification, detection, and segmentation tasks. The different tasks require a diverse CNN structure, in specific diverse model output dimensions. A categorization approach is made in figure 2.19.

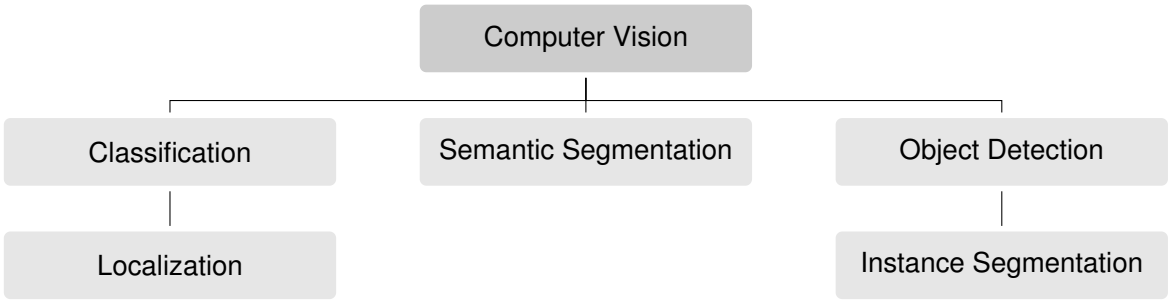

Figure 2.19: Computer Vision tasks

The input of CNNs for computer vision tasks are images, usually by size [$width$ x $height$ x 3] with 3 channels (RGB). The separate tasks differ in the network's outputs.
Image classification is the traditional image processing task in machine learning. An input image is classified in a scheme of an arbitrary amount of classes. The output contains one vector, comprising probabilities for all classes. In a localization task, further, a localization of the classified object is made. A localization is a kind of regression (2.3.2), whereby coordinates of bounding boxes for the object are computed. A classification plus localization model thus produces a classification probability with in addition bounding box coordinates as an output. A semantic segmentation model computes a classification for all pixels of the image and segments the whole image in different predefined classes. A segmentation model outputs a mask by the same size as the input image [$width$ x $height$ x 1] (with a depth of 1, as one mask is produced). Object detection and instance segmentation tasks localize and segment various objects in images, implemented by an image fragmentation during the computation process, which is further explained in 2.6.3.

2.6.2 Architectures

LeNet

In the field of Computer Vision, there are several common CNN architectures. A key progress definitely was the document recognition CNN from LeCun et al. in 1998 (LeCun et al., 1998): As a first of its kind, a CNN with gradient-based learning and computation of image data as an input was successfully applied. Figure 2.20 shows the architecture of the developed LeNet-5. In this approach, input data in form of a [32x32] pixel sized gray scaled image gets computed by a CNN with five hidden layers. In the first Conv layer C1, six feature maps are produced by [5x5] filters, which get subsequently pooled by Subsampling layer S2. In the following layers, several feature maps get computed, that decrease by dimension and increase in quantity. At the end of the CNN, a FC layer combines the information and with the execution of an Euclidean Radial Basis Function (RBF) finally the possible output predictions are made (the RBF computes radial distances of all possible classes to the predictions and decides by means of the lowest difference).

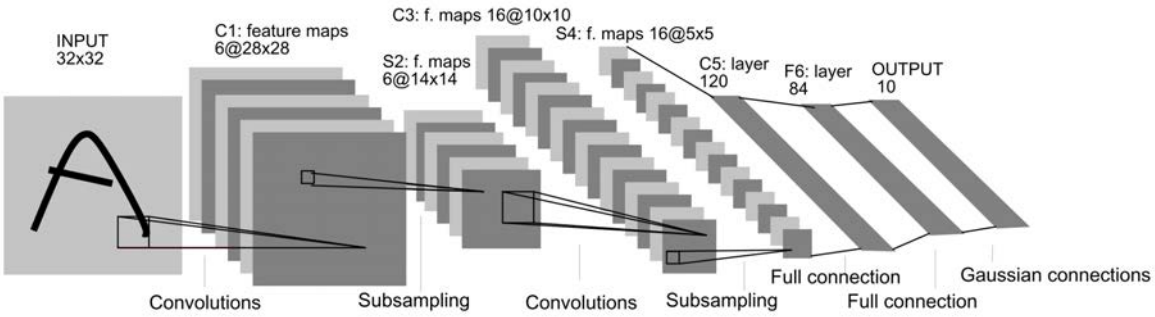

Figure 2.20: Architecture of LeNet-5, from (LeCun et al., 1998)

In training mode, the MSE function was used to compute the loss. The LeNet-5 was trained in 20 epochs on a dataset containing 60.000 training images and 10.000 test images and achieved a test error rate of 0.8%.

The development of CNN architectures can be seen on the basis of image processing results based on datasets like the CIFAR-100 (Krizhevsky, 2009), MS COCO (Lin et al., 2015), PASCAL Visual Object Classes (VOC) (Everingham et al., 2014) or the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015). The ILSVRC is an annual competition of image classification at larger scales, based on the ImageNet dataset, which contains about 1.5 million labeled images and consists of different challenge tasks like classification, object localization and object detection.
Figure 2.21 shows the top 5 error rates (error rate for a classification task its target is in top 5 predictions) of the winning networks on ILSVRC through the years. The plot vividly demonstrates that error rates decreased over the years as the networks increased in depth.

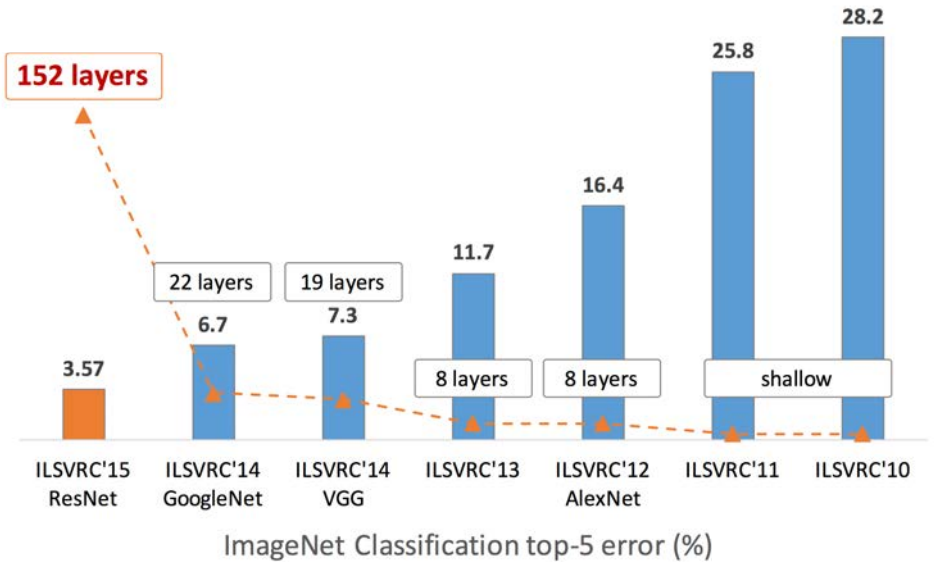

Figure 2.21: ILSVRC top-5 winning error rates, from (He et al., 2016)

AlexNet

One of the first Convolutional networks for Computer Vision tasks was the AlexNet (Krizhevsky et al., 2012), designed by the Supervision Group. In 2012, the AlexNet won the ILSVRC, outperforming the other participating networks with a top-5 test error rate of 15,3% (according to (Krizhevsky et al., 2012)), compared to the runner-up with a rate of 26,2%. The architecture of the AlexNet was similar to the LeNet, even though in addition deeper and bigger, containing 7 hidden layers, about 650.000 neurons and 6 million parameters. The AlexNet was one of the first networks stringing together Conv layers directly, instead of alternating Conv layers with Pooling layers.

GoogLeNet

The ILSVRC 2014 winner was the GoogLeNet (Szegedy et al., 2015), also known as Inception-v1, designed by Szegedy et. al from Google, a network that also used a CNN inspired by the LeNet. Its main contribution was the development of an Inception Module, which increased the depth and width of the network by simultaneous decreasing the number of parameters.

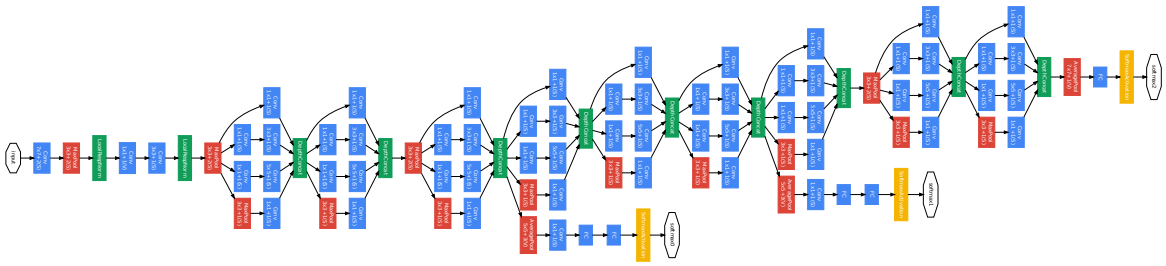

Figure 2.22: GoogLeNet, from (Szegedy et al., 2015)

An inception module, displayed in figure 2.23, is a kind of Conv layer, that computes filters of different sizes in parallel paths concatenating them at the end (Szegedy et al., 2015). Due to the concatenation of various filters, the filter sizes could decrease. This procedure and, in addition, the implementation of [1x1] convolutions before the computational expensive bigger convolutions as well as the replacement of an FC layer at the end of the network by an average pooling layer dramatically reduced the number of parameters. Compared to the AlexNet's 60 million parameters, GoogLeNet only contained about 4 million parameters, even though it was 22 layers deep. The architecture, which can be seen in figure 2.22, was composed by several Inception Modules, arranged with max pooling layers along the lines of LeNet.

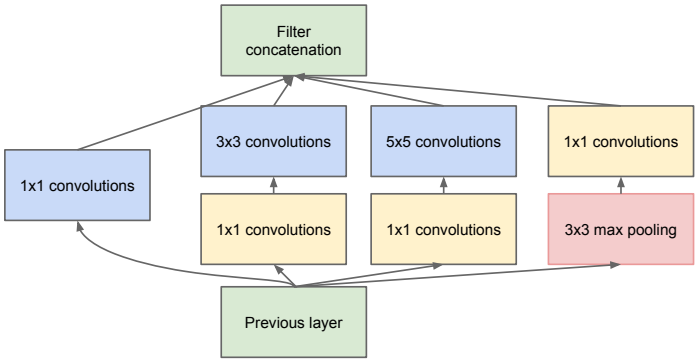

Figure 2.23: Inception Module, from (Szegedy et al., 2015)

ResNet

Residual Neural Network (ResNet) (He et al., 2016) was the winner of ILSVRC 2015. This architecture introduced so-called skip connections, as it reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Figure 2.24 shows a residual learning building block of He et al., where this method is implemented.

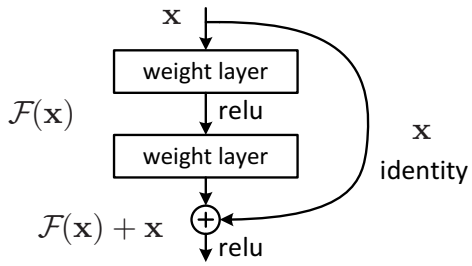

Figure 2.24: Residual learning, from (He et al., 2016)

Instead of mapping the underlying layers by the function $H(x) := F(x)$, a residual mapping of $H(x) := F(x) + x$ is implemented. The idea of ResNet is, that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping, as no more completely new parameters have to be learned but rather differences to the previous layer. "To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers" (He et al., 2016).

The implementation of a plurality of layers in ResNet as a deep CNN also became possible because of the massive implementation of a method called batch normalization (Ioffe and Szegedy, 2015). Batch normalization included layer normalization for each training mini batch in the learning process and demonstrated a way of solving problems arising by the layer normalization of Deep CNNs.
The architecture of ResNet-34, a ResNet comprising 34 layers, can be seen in figure 2.25 at the top, compared with a 34 layer plain CNN (bottom). The winning ResNet, ResNet-152, was extended to a depth of 152 layers, according to the same principle.

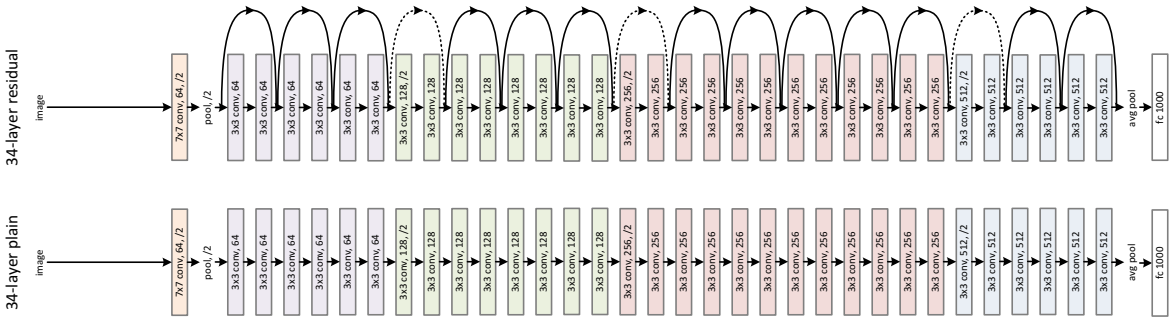

Figure 2.25: ResNet-34, from (He et al., 2016)

ResNets are currently by far state of the art CNN models and used in various networks as backbone architecture today. Latest approaches of ResNet-backboned architectures are the advanced ResNeXt (Xie et al., 2017) or the Inception-v4 (Szegedy et al., 2016), which combine the techniques of residual networks and inception modules. The winner of the latest ILSVRC in 2017 was the Squeeze-and-Excitation Network (SENet) (Hu et al., 2017), a network also based on residual learning with "Squeeze-and-Excitation" blocks, that adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels. Figure 2.26 shows an analysis of Deep Neural Networks (DNN) (Canziani et al., 2017). On the left, the networks are plotted depending on the reached top-1 score and, in addition, compared by their parameter size, displayed in size of the circles, and the amount of needed operations (right plot).

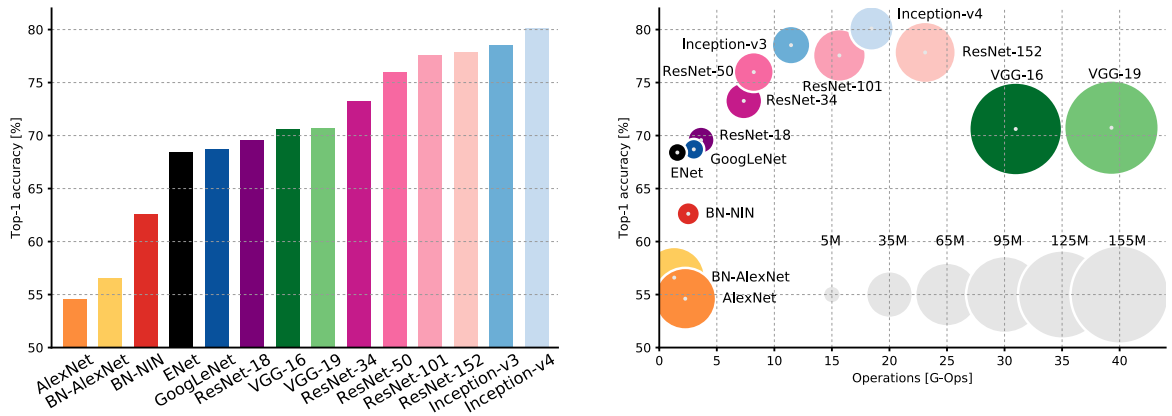

Figure 2.26: DNN Comparison, from (Canziani et al., 2017)

2.6.3 Instance Segmentation

Instance segmentation is a task whereat various objects in images are located and pixel by pixel segmented. A CNN output for an instance segmentation task therefore has to consist of a classification for the object, a bounding box computation and another classification for every pixel inside the predicted area to segment the classified object. As a CNN needs a predefined output, a prediction for various objects needs to be computed in two steps: At first, the model selects an area for every object wherein the object gets detected and segmented. In a second step, a bounding box and pixel-wise segmentation of the separate areas takes place.

R-CNN

To compute different areas for various objects, in 2012 a pioneering technique, combining region proposals with a CNN, called Regions with CNN features (R-CNN) (Girshick et al., 2012), was developed. R-CNN outperformed the method OverFeat (Sermanet et al., 2013), then state of the art, in ILSVRC 2013. It achieved a mean Average Precision (mAP) in the detection dataset of 31.4% compared to 24.3% of OverFeat.
The architecture of R-CNN was structured in four modules: The first module (figure 2.27 No. 2) generates about 2000 region proposals by means of Selective Search (Uijlings et al., 2013), a method that uses a data-driven grouping-based strategy with a variety of grouping criteria, generating high-quality object locations in any scales and aspect ratios. The regions then get processed to mean-substracted [227x227x3] images as input for the second module.

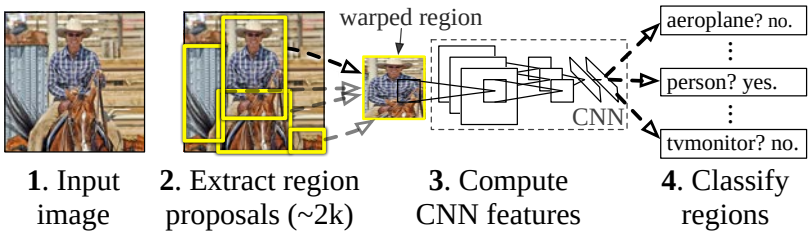

Figure 2.27: R-CNN architecture, from (Girshick et al., 2012)

The second module (figure 2.27 No. 3) is a CNN that computes the input images to a fully connected feature vector. R-CNN used a slightly modified version of the AlexNet (Krizhevsky et al., 2012) as CNN backbone architecture. The third module (figure 2.27 No. 4) is a set of class-specific linear SVMs, one per class, to classify the object in the region proposal. Finally, R-CNN runs a linear regression model as employed in DPM (Felzenszwalb et al., 2009) on the region proposals to generate bounding boxes for the object, to reduce localization errors.
As R-CNN achieved very well detection performance, however, it had notable drawbacks (Girshick, 2015): For every image with each about 2000 region proposals, thus 2000 images had to be processed. In addition, three different models (CNN, classifier and regressor) had to be trained separately. This led R-CNN to be hard and expensive to train and slow at test-time.

Fast R-CNN

Ross Girshick, first author of R-CNN, solved these problems with a further development of the algorithm in 2015, called Fast Region-based Convolutional Neural Network (Fast R-CNN) (Girshick, 2015). As the name indicates, Fast R-CNN proposed a method based on the R-CNN algorithm, improving speed and expense. Figure 2.28 illustrates the architecture of Fast R-CNN: Instead of computing various region proposals as individual images, the network processes the whole image with several Conv and Pooling layers to produce a Conv feature map. Then, a Regions of Interest Pooling Layer (RoIPool) (He et al., 2014) extracts a feature vector for each object proposal directly on the Conv feature map. This RoI feature vector subsequently gets processed by a sequence of FC layers, branched in two output layers: a softmax classification layer that classifies every class of the object, adding a "background" class as negative class, and a regression layer that outputs bounding boxes of the object. Fast R-CNN used the architectures of the AlexNet (Krizhevsky et al., 2012) from R-CNN and the VGGNet (Simonyan and Zisserman, 2015), runner-up in ILSVRC 2014, as backbone architecture.

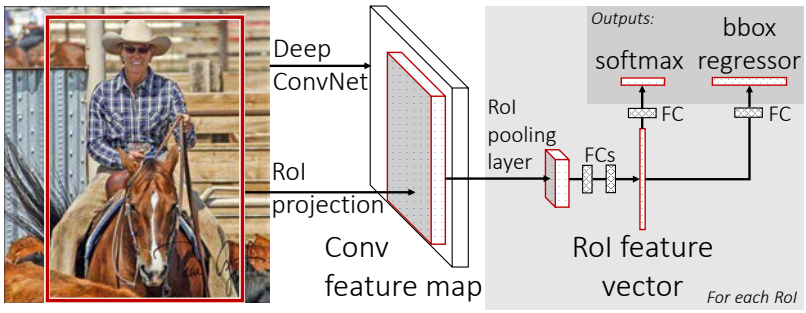

Figure 2.28: Fast R-CNN architecture, from (Girshick, 2015)

By computing the image once and applying the region proposals on the Conv feature map, instead of processing every region proposal in the CNN separately, Fast R-CNN got significantly more computationally efficient. By replacing the three individual ML models with one combined model, Fast R-CNN was in addition a lot easier to train, as the CNN, classification and regression could be adjusted together. It trained 9x faster than R-CNN, was 213x faster at test-time, actually achieving a 4% higher mAP measured on PASCAL VOC 2012 (Everingham et al., 2010) (Girshick, 2015).

Faster R-CNN

One remaining bottleneck was the region proposal generation. As Fast R-CNN achieves near real-time predictions, when ignoring the time spent on region proposals, the used method to propose object regions, Selective Search (Uijlings et al., 2013), was quite expensive (at 2 seconds per image) (Ren et al., 2017). In 2015, a team at Microsoft Research introduced a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN, a fully convolutional

network that simultaneously predicts object bounds and objectness scores at each image position, merged with Fast R-CNN for object detection to one single network was the result of their work, called Faster R-CNN (Ren et al., 2017).
Instead of running a separate Selective Search algorithm, the architecture of one single CNN including a RPN was introduced. Faster R-CNN uses the Conv feature maps, used by the region-based detector of Fast R-CNN, for generating region proposals, which is illustrated in 2.29.

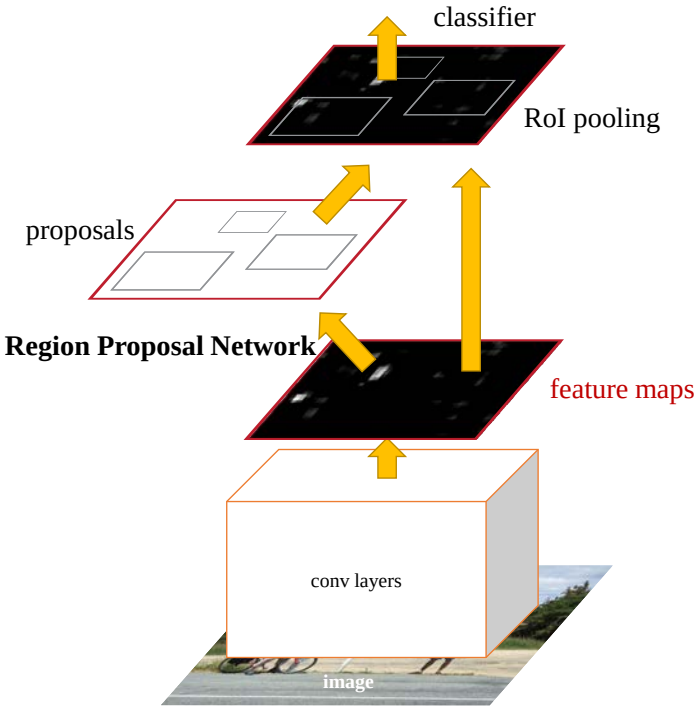

Figure 2.29: Faster R-CNN architecture, from (Ren et al., 2017)

The Regions of Interest (RoI) are generated by the RPN, that outputs bounding boxes by inputting a Conv feature map. The RPN, shown in figure 2.30, slides a spatial window by size [3x3] over the feature map, mapping it to an intermediate layer. This feature map is fed into two sibling FC layers, a bounding box regression layer (reg) and a classification layer (cls). At each sliding-window location, the RPN predicts k region proposals, thus generating an output of $2k$ classification scores (object / no object) and $4k$ regression values (bounding box coordinates). The k proposals, parameterized relative to k reference boxes, called anchors, possess 3 different scales and aspect ratios. With a given Conv feature map by size [W x H], the RPN produces $W * H * k$ anchors. Each anchor box that is due to its classification score likely to be an object then is passed into Fast R-CNN to generate a classification and tightened bounding boxes.

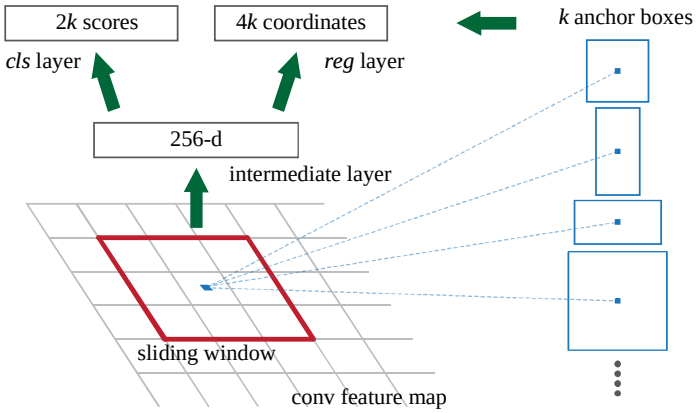

Figure 2.30: RPN architecture, from (Ren et al., 2017)

The RPN had an additional computing cost of 10 ms per image. Thus, Faster R-CNN achieved a frame rate of 5 fps on GPU for the VGGNet (Simonyan and Zisserman, 2015), although achieving state of the art mAP's on MS COCO (Lin et al., 2015) and PASCAL VOC (Everingham et al., 2014) (Ren et al., 2017).

Mask R-CNN

The combination of object detection and semantic segmentation is a pixel-wise segmentation of the located objects. This task, called instance segmentation, was explored in 2017 by Facebook AI Research (FAIR) with Mask R-CNN (He et al., 2017). Mask R-CNN extends Faster R-CNN by adding an additional branch for object mask predictions in every Region of Interest (RoI) in parallel with the branches for classification and bounding box regression.
The RoIPool of Fast R-CNN was not designed for pixel by pixel alignment between network input and output, as a pooling of the RoI in the Conv feature map leads to a misalignment during unpooling at the mask generation by performing coarse spatial quantization for feature extraction (He et al., 2017). To fix this misalignment, Mask R-CNN introduced a quantization-free layer, called RoIAlign, that preserves exact spatial locations. Instead of rounding selected image pixels in the pooled feature map, a bilinear interpolation (Jaderberg and Deepmind, 2015) to compute the exact values of the input features was done.

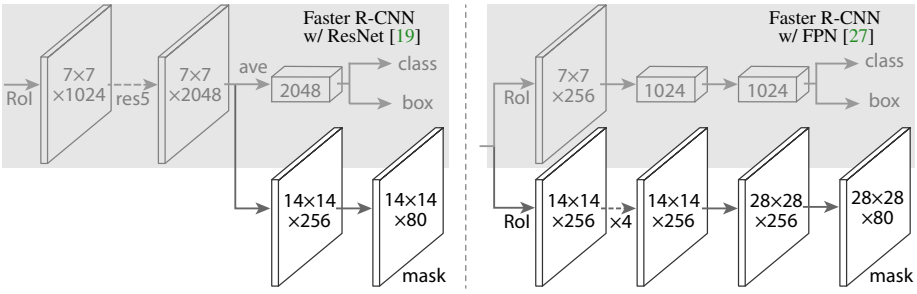

Figure 2.31: Mask R-CNN architecture, from (He et al., 2017)

The design of Mask R-CNN is illustrated in 2.31. The backbone architecture of Mask R-CNN is a combination of the ResNet (He et al., 2016) and a Feature Pyramid Network (FPN) (Lin et al., 2017), an extension that better represents objects at multiple scales. Figure 2.32 shows the principle of the FPN, which uses a top-down architecture with lateral connections to build an in-network feature pyramid from a single-scale input, thus extracting RoI features from different levels of the feature pyramid according to their scale, instead of extracting them from one feature map. On top of the backbone architecture, the branches for classification, bounding box regression and mask generation are added (figure 2.31).

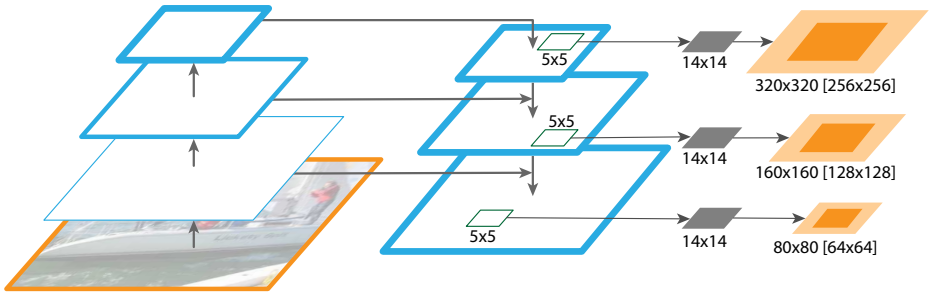

Figure 2.32: FPN architecture, from (He et al., 2017)

Mask R-CNN outperformed all single-model entries on instance segmentation, bounding box object detection and person keypoint detection tasks of the MS COCO challenges (Lin et al., 2015). Today, it is kind of state of the art CNN architecture for instance segmentation tasks.

Chapter 3

Dataset

The Mask R-CNN requires pre-labeled visual input data like photos or videos. As the implementation is made for an instance segmentation of construction elements, the applied dataset consists of a set of construction site photos. The network trains on 747 photographs of different time steps at various construction sites. The images are for the most part captured as aerial photographs by UAVs, complemented by some photos captured by cameras from the ground. Two exemplary extracts of the dataset can be seen in figure 3.1.

Figure 3.1: Aerial (left) and from ground taken (right) construction site photo

The image size of the used photos is in a range of [4000x3000] to [6000x4000] pixels. The network processes all images as JPEG files by size [2048x2048]. To reduce overfitting, the dataset images are multiplied by augmentation to a number of 4482 images (see 4.2). The dataset is split into a training set, a validation set and a test set at the ratio of approx. 75%, 15% to 10%. The model trains on the training set, containing 3376 images, and on 672 validation images, whereas the segmentation results are evaluated on the test set, consisting of 434 images. In the course of further development of the construction site instance segmentation tool, the dataset will continuously be enlarged.

Chapter 4

Methods

4.1 Workflow

The main purpose of this research is the development of a construction site object segmentation tool. It is realized as an implementation of the Mask R-CNN (He et al., 2017), applied on construction site photographies. The development of the tool is structured in six work steps and can be seen in figure 4.1, whereas the four dark gray colored steps constitute the actual machine learning application.

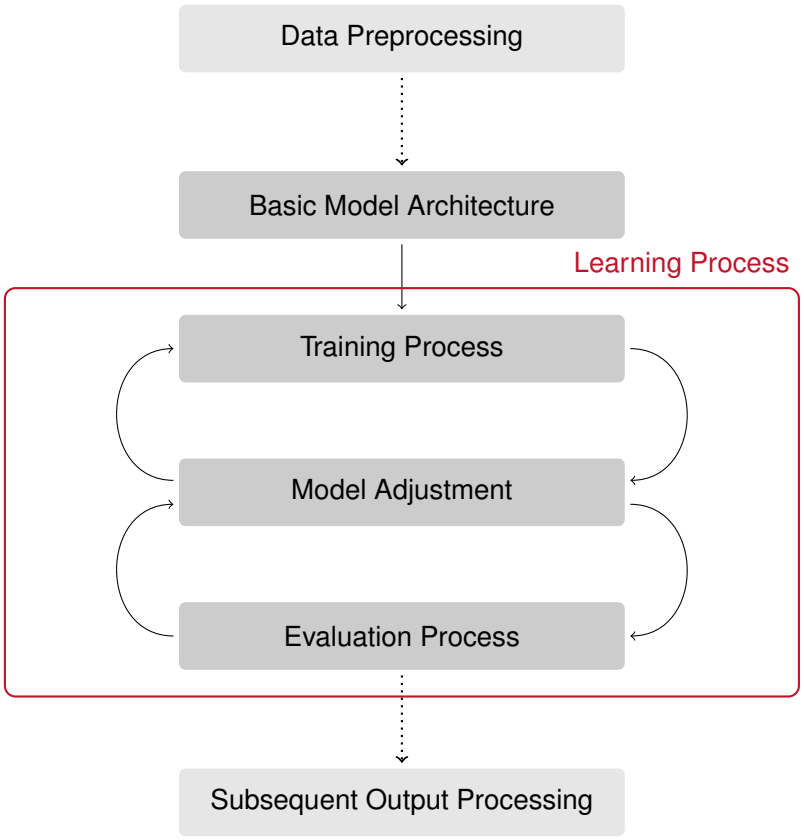

Figure 4.1: Tool development workflow

4.2 Data Preprocessing

The input of the network includes two types of data: images to be learned from as well as mask coordinates of desired labeled objects in the image. The data preprocessing task includes the production and preparation of the input data for the implementation in Mask R-CNN.

Labeling

For the learning process, the CNN needs a dataset comprizing images with prelabeled and pre-segmented objects as input data. For image labeling, a variety of tools are available and can be accessed free of charge. The input dataset (described in 3) is labeled by use of the online labeling platform Labelbox. As the first stage of development, the instance segmentation is in the course of this thesis applied to formwork elements. Therefore, all formwork elements in the photos are labeled, realized by separate polygons. For instance, one labeled training image can be seen in figure 4.2. The coordinates of the created mask polygon labels are saved in the WKT markup language format as JSON files for further processing.

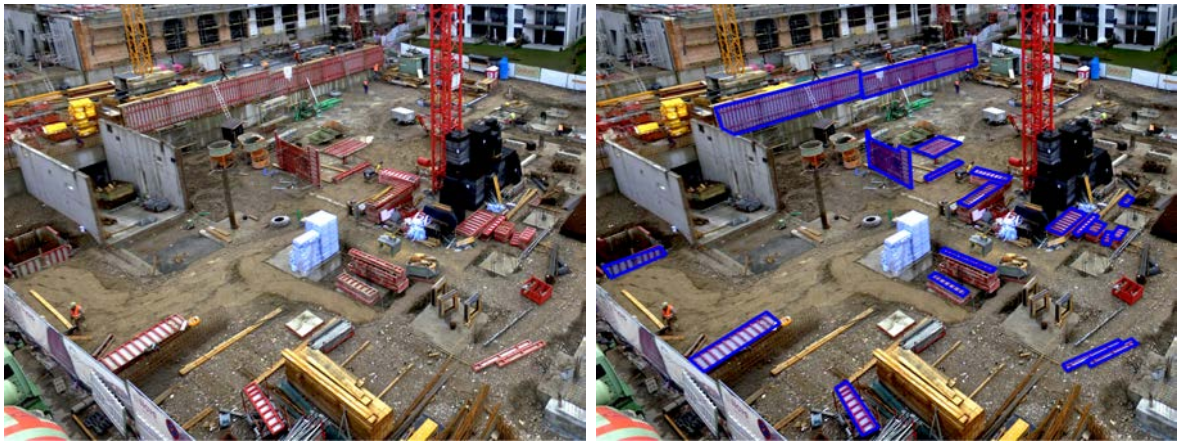

Figure 4.2: unlabeled (left) and labeled (right) construction site photo

Label Processing

The annotations of the labels then get further processed and adjusted. Labels accidentally containing only one or two coordinates are unusable, as with these labels no surface can be spanned. The ”defect” labels, as well as annotation entrys of unlabeled images, get filtered out of the produced JSON file by means of RegEx operations. For the Mask R-CNN, the MS COCO (Lin et al., 2015) data format is used as label input data format. Inputting the labels in this data format enables an inclusion of the MS COCO datasets to the dataset without much effort, if necessary for further tasks. Consequently, as the last step, a modification of the data format takes place.

Augmentation

To avoid overfitting effects, during the training process the dataset is increased by use of data augmentation techniques. For the creation of new training examples, a series of image processing tools is applied to the images. The images are augmented in such a manner, as other photos under real conditions could appear, to generate realistic and useful instances and allow more accurate predictions on new images. The used modification techniques are image flips, image distortion and adjustment of brightness and contrast. The dataset is multiplied by a factor of six, thus generating five additional training examples from one image. Every construction site photo gets distorted in width by a ratio of 0.8 as well as 1.2, whereas the scaling simulates photos captured under more respectively less acute perspectives. As the CNN interprets images as NumPy arrays comprizing single digits, such modifications, that change the values of adjacent pixels, can be used to generate largely independent new training examples. All three variations then are flipped horizontally, thus doubling the images. The produced images subsequently are modified in brightness and contrast by multiplication with randomly generated factors, in a range of 0.6 to 1.4 with regard to the initial value, imitating different lighting conditions. The generated instances of one training example can exemplarily be seen in figure 4.3.

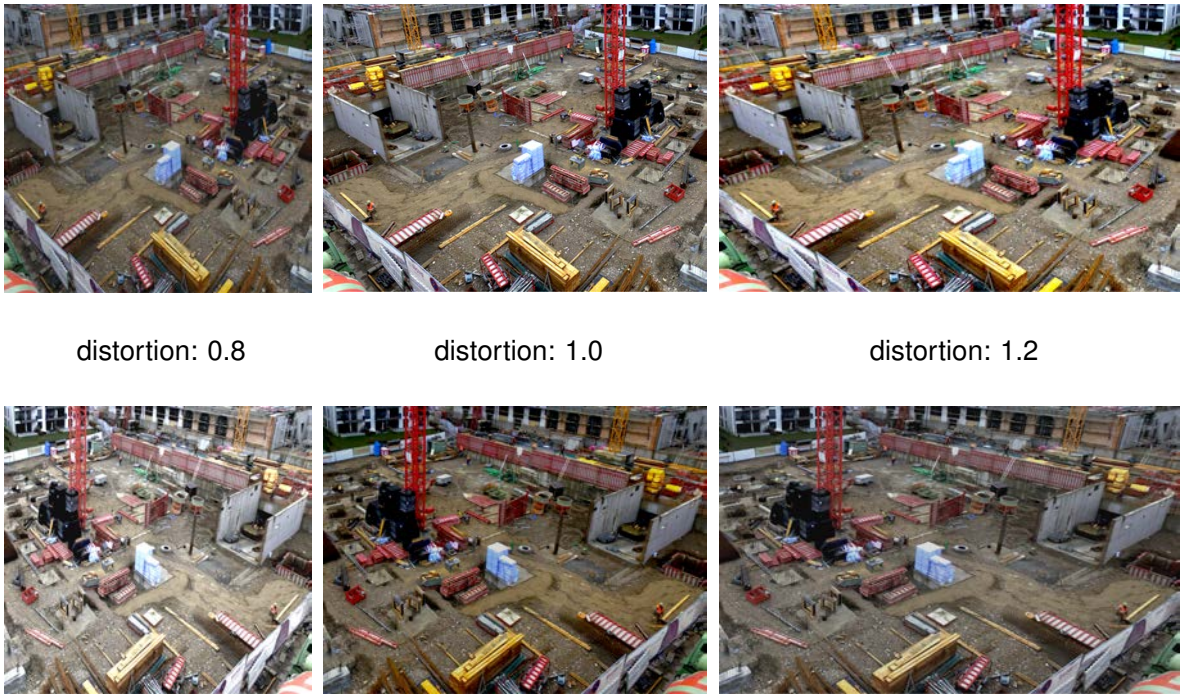

Figure 4.3: Augmentated images in initial orientation (top) and horizontally flipped (bottom)

Partitioning

Finally the dataset, i.e. the images inclusive their related annotations, are randomly partitioned into a training set, a validation set and a test set. The split is made in a relation of approximately 75% training data, 15% validation data and 10% test data.

4.3 Basic Model Architecture

The task of this thesis is the development of a pixel accurate tool to select construction elements and equipment. As a typical instance segmentation challenge, a CNN for instance segmentation is applied. Today, the Mask R-CNN (He et al., 2017) outperforms other segmentation network architectures (Chen et al., 2018), achieving higher Average Precisions (AP's) than for example the winners of MS COCO challenge (Lin et al., 2015). As state of the art network architecture, the Mask R-CNN is implemented, whereas a description of the theoretical construction and the model structure is given in 2.6.3. For this implementation, a ResNet-101 (He et al., 2016) and a FPN (Lin et al., 2017) backbone architecture are used, based on a TensorFlow backend. The configurations for this project relate to the base configurations used to train the MS COCO dataset, apart from a few adaptions. An extract from the used configurations can be seen in figure 4.4.

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     2
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.80
GPU_COUNT                      1
IMAGES_PER_GPU                 2
IMAGE_MAX_DIM                  2048
IMAGE_MIN_DIM                  2048
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [2048 2048 3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
MINI_MASK_SHAPE                (56, 56)
NUM_CLASSES                    2
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (64, 128, 256, 512, 1024)
STEPS_PER_EPOCH                1700
TRAIN_ROIS_PER_IMAGE           256
USE_MINI_MASK                  True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Figure 4.4: Extract of the Mask R-CNN configuration

Depending on the capability of the system the training was executed on (see 4.4), a different amount of images per GPU and GPUs working parallel was used. The computed batch size results by multiplication of the number of GPUs with the number of images per GPU. The maximal number of instances, detected in the images, is set to 100, whereas the threshold of confidence for detections is set to 80%. Given the fact, that the processed images were captured at long distances, thus containing a lot of image content of small size, the photos had to be computed by large resolution. As a compromise of short training durations (see 4.4) and minimal information loss, the configuration uses a processing resolution by size [2048x2048]. All images are adapted into this resolution by size correction and zero padding to a quadratic size. The implementation uses a learning rate of 0.001, a momentum of 0.9 and weight decay regularization of 0.0001. To reduce the training durations, it computes the masks as mini masks of shape [56x56]. The amount of implemented classes is set to two, containing one class for formwork elements and a background class. During the training process, the model produces a number of 256 RoI's with a positive ratio of approximately 33%. The RPN creates anchors with ratios of 0.5, 1.0 and 2.0, in scales between 64 and 1024 pixels. The training epochs include one-time processing of all training images, thus the steps per epoch depend on the size of the training set. The number of computed images per epoch is calculated by multiplication of steps per epoch and the batch size. The number of images computed for validation after every epoch is set to 100.

4.4 Training Process

Transfer Learning

Starting from weights pre-trained on other images can dramatically decrease the required amount of training data, as the pre-trained weights have already learned common image features (Pan and Yang, 2010). As the available dataset only contains about 750 images, the model uses this technique, called transfer learning, for training. The tool trained starting with weights pre-trained on the MS COCO dataset (Lin et al., 2015). This dataset has a volume of approximately 200.000 labeled images with 80 object categories, that, not even containing construction site photos or classes, though worked very well as origin.

Training Epochs

The pre-trained weights in the deeper layers of the network already fit relatively well and merely have to be finetuned. However, the head network layers have to be adjusted more thorough, resulting in a model that trains in three stages: in stage one, the network heads are trained to an extent up to 50 epochs. In every epoch, every image of the training set gets processed once. For one training epoch with a batch size of two and the augmented training set comprizing narrowly 3400 images, hence in one epoch, 1700 steps get computed. In stage two, all layers from ResNet stage 4 and up get finetuned in 30 epochs. Finally, the model finetunes all layers corporately for another 30 epochs, using one-tenth of the learning rate.

Timing

The Mask R-CNN implementation with ResNet-101 and FPN backbone architecture is a fairly large model and needs high-performance GPU computing power. The computation of one photo in a [2048x2048] resolution approximately needs a GPU with at least 8 GB of RAM. An evaluation of the training runtimes on different NVIDIA GPUs and in addition on a CPU can be seen in figure 4.5, whereas because of a lack of memory the runtime analysis on the NVIDIA GeForce 660Ti and NVIDIA Quadro P2000 was executed with smaller image resolutions. The model mainly trained on an NVIDIA Tesla P100 GPU, respectively an NVIDIA DGX-1 system, comprizing 8 Tesla P100 GPUs. In inference mode, the model runs at ∼500ms on an NVIDIA Tesla P100 GPU.

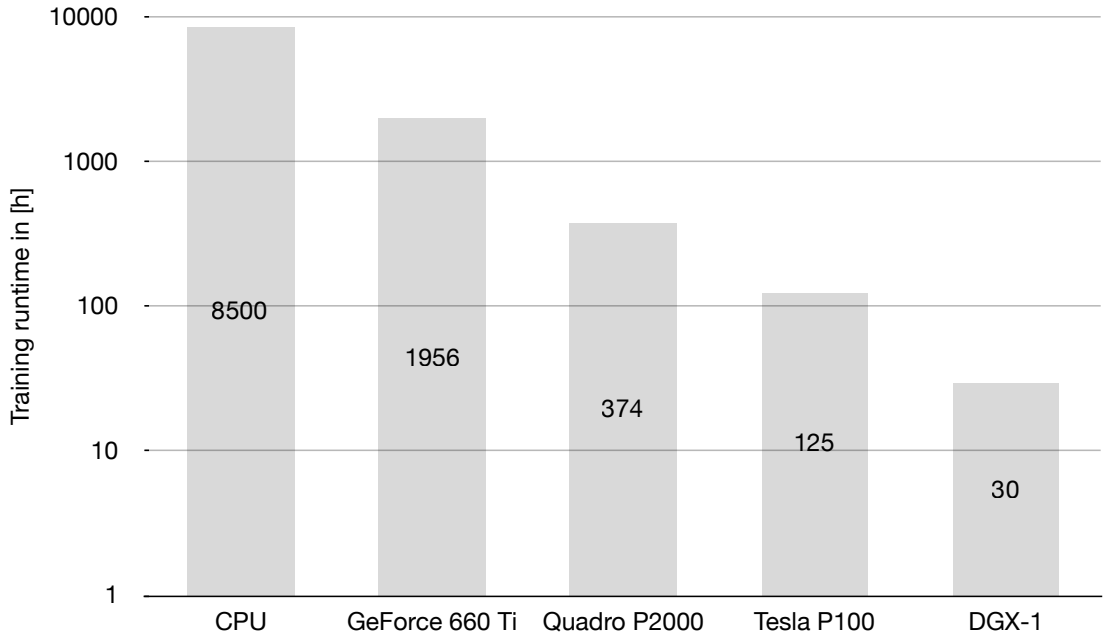

Figure 4.5: Training runtime analysis

4.5 Model Adjustment and Evaluation

Training process

The first hyperparameter to be set was the learning rate. The learning rate of the COCO config (4.3) performed well and was adopted for further training, whereas after every adjustment of the dataset its functionality was verified. Using higher computing capability GPUs, an increase of the image processing resolution from initial [800x800] to [2048x2048] pixels was possible, maintaining moderate runtimes. During the training process, an analysis of the training and validation loss curves visualized at an early stage, that with the initial amount of approximately 750 images, in the relatively deep model overfitting effects appeared. A typical indication for overfitting, a large gap between the training loss and the validation loss as a function of the training progress, can be seen with the plotted error curves in figure 4.6.

To minimize overfitting, the dataset size thereafter was, as a first step, quadrupled and then sextupled. The summed training and validation loss of classification, bounding box and mask generation, for comparing the three different dataset sizes, are (smoothed out) plotted as a function of the training epochs in figure 4.6, whereas the loss curves during 50 epochs of network head training are shown. The dataset enlargement by means of augmentation reduced the gaps between training and validation loss curves by 70%, thus almost halving the absolute value of the validation losses.

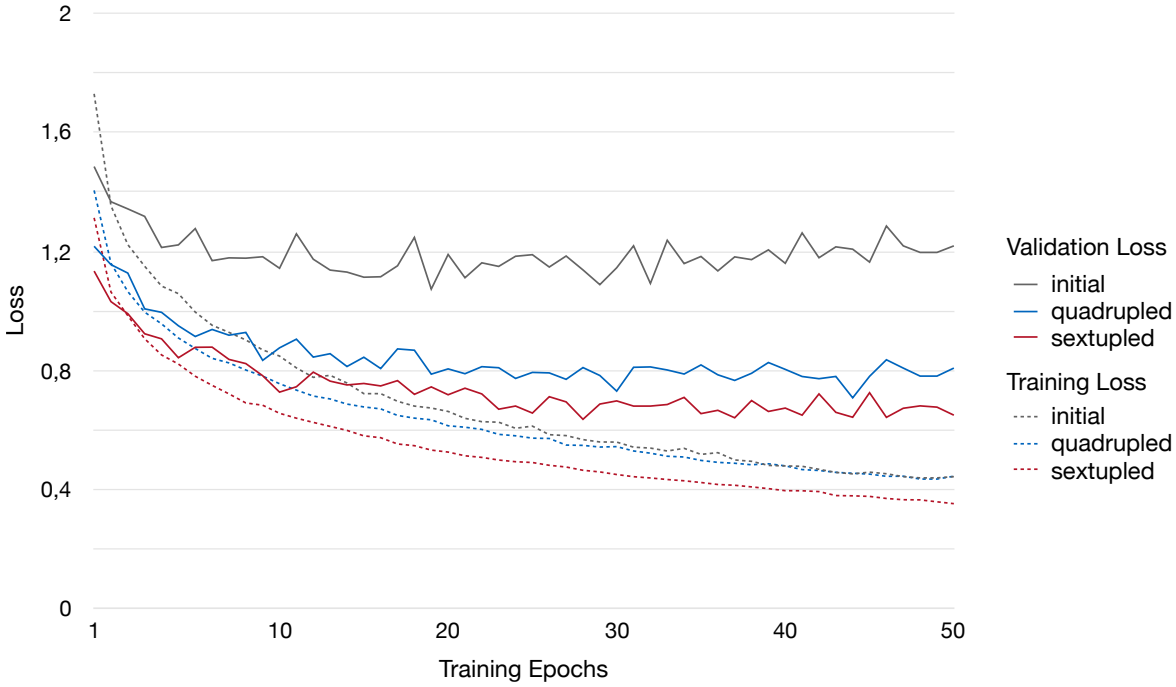

Figure 4.6: Loss curves over the network head training epochs

Figure 4.6 also demonstrates that the augmentation forfeits efficiency with increasing extent. Considering, that with an increasing dataset size the runtime extends, the best cost-benefit ratio of augmentation has to be assessed. Figure 4.7 shows an approximation of the loss trend as well as the runtime increase as a function of the dataset size multiplication by augmentation. Taking this function into account, the dataset was not further augmented. However, it has to be mentioned, that the approximation function of the validation loss trend is quite vague, as it is derived only out of the averages of three tested dataset augmentation factors. For further improvement of accuracy, an inclusion of new images is recommended.

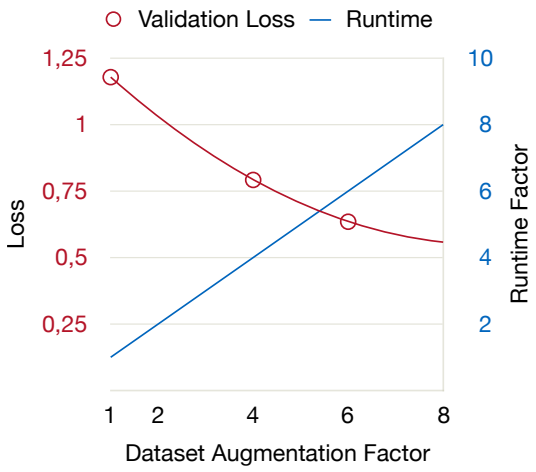

Figure 4.7: Loss vs. runtime

Final Results

The validation of the training process takes place by use of the training and validation sets. The error results, however, get evaluated on the test set. The test set (see 3) contains 434 images and gets processed for evaluation after the training runs. The common metric to measure the accuracy of object detection and instance segmentation models is the mAP score, which is evaluated for this implementation. The mAP can be interpreted as the average over all classes, of the precisions at different recall values, and is an extension of the AP. For every image, the recall and precision value is calculated. The mAP then sums up precisions over the various recalls and averages them over the test set.
Considering the values of precision and recall, it can be ascertained that the mean recall value is lower than the model's precision. As the model's accuracy depends on the quality of the input data, the fact that the recall, in other words, the ability of the model to identify all correct instances, achieves poor values, may be a result of the lack of labeling of occasional objects. As an analysis of the labeled data supports this presumption, a thorough revision of the labeled data is proposed to improve further accuracy.
The mAP of this tool increased with augmentation of the dataset and optimization during the training process. Without augmentation, the mAP reached a maximal value of 0.66, using the described configurations. Augmenting the dataset, the model in its first stage of development finally achieved an mAP of 0.91 on the test set.

4.6 Subsequent Output Processing

For the application of the image segmentation tool, the predictions of the model can be processed and saved in different data formats. In the course of the automated construction progress monitoring tool progressTrack (Braun et al., 2018), the model outputs the predicted results in three ways: The bounding boxes and prediction probabilities of the objects in the processed images are written into a JSON file. The predicted masks are saved as NumPy binary files and optically displayed in a PNG image file.

Chapter 5

Discussion

The evaluation of the model shows acceptable results for the instance segmentation tool in this first development stage to be employed. A further improvement and extension of the tool is planned and will be implemented in the near future. The next strategic objectives to be pursued are the upgrading to segment further construction elements and an enhancement of the use of information that can be gained by processing the images. At an operating level, this will be implemented with a revision and enlargement of the dataset, including a labeling of other elements and corresponding model training with the improved data. Further output processing operations and investigations of the practicality are planned to extend the range of application possibilities.

It may be noted here that, even apart from object detection and computer vision, machine learning is a very powerful tool to perform a variety of tasks and relatively easy to implement, actually without much prior knowledge. If the development continues in the way as it has been doing over the last years, which can be expected, a plurality of jobs, today exercised by a human, will in the not too distant future be performed by computer applications. This evolution will also take place in civil engineering, whereas applications like automated monitoring or crack detection tools merely represent the beginning.

Appendix A

Glossary of Terms

The machine learning terms defined in this glossary are, unless otherwise stated, derived and enhanced based on the Google Developers Machine Learning Glossary (Google Developers, 2018).

accuracy

The fraction of correct predictions in a classification model, defined as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total number of examples}}$$

activation function

A function that takes in the weighted sum of all of the inputs from the previous layer and then generates and passes an output value to the next layer. Typically used activation functions are ReLU, the sigmoid or the tanh function.

backpropagation

The primary algorithm for performing gradient descent on neural networks. First, the output values of each node are calculated (and cached) in a forward pass. Then, the partial derivative of the error with respect to each parameter is calculated in a backward pass through the graph.

batch

The set of examples used in one iteration (one gradient update) of model training.
The training set of preprocessed data is split into several batches.

bias

An intercept or offset from an origin. A bias term in machine learning models is referred to an initial value in each layer of the network.

convolutional layer

A layer of a neural network in which a convolutional filter passes along an input matrix. The convolutional layers process input features, learning optimal weights by means of backpropagation.

dataset

The collection of (preprepared) examples used to train and validate the ML model.

decision boundary

The separator between classes learned by a model in binary class or multi-class classification problems. During the training process, the decision boundary best fitting to the according problem is evaluated.

feature

An input variable used in making predictions. During development of an ML model, different input features are tested, evaluated and combined to find most effective combinations.

fully connected layer

A layer in a CNN that is fully connected to the previous layer, which means that every neuron in this layer is connected to every neuron in the previous layer. A FC layer typically is located at the end of a CNN.

gradient descent

A technique to minimize loss by computing the gradients of loss with respect to the model's parameters, conditioned on training data. Informally, gradient descent iteratively adjusts parameters, gradually finding the best combination of weights and bias to minimize loss.

hidden layer

A synthetic layer in a neural network between the input layer (the features) and the output layer (the prediction). A neural network contains one or more hidden layers.

hyperparameter

Parameters of the ML model that are optimized for the best prediciton results. Hyperparameters are for example the Learning Rate (LR) or the model architecture.

learning rate

A scalar used to train a model via gradient descent. During each iteration, the gradient descent algorithm multiplies the learning rate by the gradient. The resulting product is called the gradient step.

loss

A measure of the distance of model's predictions to its label. Typically used loss functions are for example MSE for linear regression models or Log Loss for logistic regression models.

neuron

A node in a neural network, typically taking in multiple input values and generating one output value. The neuron calculates the output value by applying an activation function to a weighted sum of input values.

normalization

The process of converting an actual range of values (through division and substraction) into a standard range of values, typically [-1, 1] or [0, 1].

one vs. all

Given a classification problem with n different classes, a one-vs.-all solution consists of n separate binary classifiers for each class.

overfitting

Phenomenon of a model fitting the training data so closely, that the decision boundaries no longer map correct predictions on new data.

pooling

Pooling, less formally called *subsampling* or *downsampling*, is an operation reducing a matrix created by earlier layers to a smaller matrix.

precision

Precision identifies the frequency with which a model was correct when predicting the positive class, defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

recall

Recall presents the frequency, how many possible positive labels the model correctly identified, defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

Rectified Linear Unit (ReLU)

An activation function by formula:

$$f(x) = max(0, x)$$

regularization

The penalty on a model's complexity, regularization helps prevent overfitting.

sigmoid function

A function that maps logistic or multinomial regression output to probabilities, returning a value between 0 and 1. The sigmoid function is defined as:

$$y = \frac{1}{1 + e^{-\sigma}}$$

test set

A subset of the data set that you use to test a ML model after the model has gone through initial vetting by the validation set.
In combination with training set and validation set part of the dataset.

training set

A subset of data used to train a ML model.
In combination with validation set and test set part of the dataset.

validation set

A subset of data - disjunct from the training set - that is used to adjust hyperparameters.
In combination with training set and test set part of the dataset.

weight

A coefficient for a feature in a linear model, or an edge in a deep network. The goal of training a linear model is to determine the ideal weights for each feature. If a weight is 0, then its corresponding feature does not contribute to the model.

Bibliography

Arnab, A. and P. H. Torr (2017, jul). Pixelwise instance segmentation with a dynamically instantiated network. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Volume 2017, pp. 879–888. IEEE.

Bai, M. and R. Urtasun (2017, jul). Deep watershed transform for instance segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Volume 2017, pp. 2858–2866. IEEE.

Boureau, Y.-L., J. Ponce, and Y. LeCun (2010). A theoretical analysis of feature pooling in visual recognition. *Proceedings of the 27th international conference on machine learning (ICML-10)* (December 2016), 111–118.

Bradski, G., A. Kaehler, and G. Bradski (2013). *Learning OpenCV: Computer vision with the OpenCV library*, Volume 53.

Braun, A., S. Tuttas, A. Borrmann, and U. Stilla (2015). Automated progress monitoring based on photogrammetric point clouds and precedence relationship graphs. In *Proceedings of the 32nd International Symposium on Automation and Robotics in Construction and Mining*, pp. 274–280.

Braun, A., S. Tuttas, U. Stilla, and A. Borrmann (2018). Process- and computer vision-based detection of as-built components on construction sites. In *Proceedings of the 35th International Symposium on Automation and Robotics in Construction and Mining*.

Canziani, A., A. Paszke, and E. Culurciello (2017, may). An Analysis of Deep Neural Network Models for Practical Applications. *arXiv*, 1–7.

Chen, L. C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence 40*(4), 834–848.

Chi, S. and C. H. Caldas (2011, jul). Automated Object Identification Using Optical Video Cameras on Construction Sites. *Computer-Aided Civil and Infrastructure Engineering 26*(5), 368–380.

Cho, C., J. Park, K. Kim, and S. Sakhakarmi (2018). Machine Learning for Assessing Real-Time Safety Conditions of Scaffolds. In *35th International Symposium on Automation and Robotics in Construction (ISARC 2018)*.

Dai, J., Y. Li, K. He, and J. Sun (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks.

Everingham, M., S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2014). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision 111*(1), 98–136.

Everingham, M., L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision 88*(2), 303–338.

Fang, Q., H. Li, X. Luo, L. Ding, H. Luo, T. M. Rose, and W. An (2018, jan). Detecting non-hardhat-use by a deep learning method from far-field surveillance videos. *Automation in Construction 85*, 1–9.

Farabet, C., C. Couprie, L. Najman, and Y. LeCun (2013). Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*(8).

Felzenszwalb, P. F., R. B. Girshick, D. Mcallester, and D. Ramanan (2009). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*(9), 1–20.

Gil, D., G. Lee, and K. Jeon (2018). Classification of Images from Construction Sites Using a Deep-Learning Algorithm. *Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC 2018)*.

Girshick, R. (2015, dec). Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, Volume 2015 Inter, pp. 1440–1448. IEEE.

Girshick, R., J. Donahue, T. Darrell, and J. Malik (2012). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. *PMLR 9*, 249–256.

Google Developers (2018). Machine Learning Glossary. In *Machine Learning Glossary*. https://developers.google.com/machine-learning/glossary/ (accessed 2018-05-31).

Ha, I., H. Kim, S. Park, and H. Kim (2018). Image-based Indoor Localization using BIM and Features of CNN. *Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC 2018)*.

Hagan, M. T., H. B. Demuth, M. H. Beale, and O. D. Jess (1996). *Neural Network Design*, Volume 2. PWS Publishing Co.

Hamledari, H., B. McCabe, and S. Davari (2017, feb). Automated computer vision-based detection of components of under-construction indoor partitions. *Automation in Construction 74*, 78–94.

He, K., G. Gkioxari, P. Dollar, and R. Girshick (2017). Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, Volume 2017-October, pp. 2980–2988.

He, K., X. Zhang, S. Ren, and J. Sun (2014). SPP: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *European Conference on Computer Vision (ECCV)*, 1–14.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hu, J., L. Shen, and G. Sun (2017, sep). Squeeze-and-Excitation Networks. *arXiv*.

Ioffe, S. and C. Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

Jacobs, R. A. (1988, jan). Increased rates of convergence through learning rate adaptation. *Neural Networks 1*(4), 295–307.

Jaderberg, M. and G. Deepmind (2015). Spatial Transformer Networks. *arXiv*, 1–14.

Karpathy, A., F.-F. Li, and J. Johnson (2016). CS231n: Convolutional Neural Networks for Visual Recognition. *CS231n Convolutional Neural Networks for Visual Recognition*, 1–2.

Kasneci, E., G. Kasneci, C. K. Thomas, and W. Rosenstiel (1997). *Artificial Neural Networks*, Volume 4.

Kim, C., B. Kim, and H. Kim (2013, nov). 4D CAD model updating using image processing-based construction progress monitoring. *Automation in Construction 35*, 44–52.

Kim, H., H. Kim, Y. W. Hong, and H. Byun (2018). Detecting Construction Equipment Using a Region-Based Fully Convolutional Network and Transfer Learning. *Journal of Computing in Civil Engineering 32*(2), 04017082.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. *Science Department, University of Toronto, Tech.*, 1–60.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 1–9.

LeCun, Y. and Y. Bengio (1995). Convolutional Networks for Images, Speech, and Time-Series. *The handbook of brain theory and neural networks*.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2323.

Li, Y., H. Qi, J. Dai, X. Ji, and Y. Wei (2017, nov). Fully convolutional instance-aware semantic segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Volume 2017-Janua, pp. 4438–4446.

Lin, G., A. Milan, C. Shen, and I. Reid (2017). RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Volume 2017-Janua, pp. 5168–5177.

Lin, T.-Y., C. L. Zitnick, and P. Doll (2015). Microsoft COCO : Common Objects in Context. pp. 1–15.

Long, J., E. Shelhamer, and T. Darrell (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Volume 07-12-June, pp. 3431–3440.

MathWorks (2016). *What is Machine Learning?* https://de.mathworks.com/content/dam/mathworks/tag-team/Objects/i/88174_92991v00_machine_learning_section1_ebook.pdf (accessed 2018-04-05).

Michalski, R. S., J. G. Carbonell, and T. M. Mitchell (Eds.) (1983). *Machine Learning - An Artificial Intelligence Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Nair, V. and G. E. Hinton (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning* (3), 807–814.

Ng, A. (2000). Margins: Intuition. *Intelligent Systems and their Applications IEEE pt.1*(x), 1–25.

Ng, A. (2012). 1. Supervised learning. In *CS229: Machine Learning, Stanford University*, Volume 1, pp. 1–30.

Ng, A. (2017). Deep Learning. In *CS229: Machine Learning, Stanford University*, Volume 1, pp. 1–3.

Ng, A. (2018). Machine Learning: Regularized Linear Regression. In *Machine Learning: Lecture notes, Coursera*. https://www.coursera.org/learn/machine-learning/supplement/pKAsc/regularized-linear-regression (accessed 2018-06-10).

Pan, S. J. and Q. Yang (2010, oct). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering 22*(10), 1345–1359.

Pinheiro, P. and R. Collobert (2014). Recurrent convolutional neural networks for scene labeling. *Proceedings of the 31st International Conference on International Conference on Machine Learning 32*(June), 82–90.

Ren, S., K. He, R. Girshick, and J. Sun (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 39*(6), 1137–1149.

Ronneberger, O., P. Fischer, and T. Brox (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer Assisted Intervention - MICCAI 2015*, pp. 234–241.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*(3), 211–252.

Samuel, A. L. (1959, jul). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development 3*(3), 210–229.

Schölkopf, B. (1998, jul). SVMs - A practical consequence of learning theory. *IEEE Intelligent Systems and Their Applications 13*(4), 18–21.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (2013). OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. pp. 1312.6229.

Simonyan, K. and A. Zisserman (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations (ICRL)*, 1–14.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15*, 1929–1958.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* (2010), 8609–8613.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. Alemi (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015, jun). Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Volume 07-12-June, pp. 1–9. IEEE.

Uijlings, J. R., K. E. Van De Sande, T. Gevers, and A. W. Smeulders (2013). Selective search for object recognition. *International Journal of Computer Vision 104*(2), 154–171.

Wu, Y., H. Kim, C. Kim, and S. H. Han (2010). Object Recognition in Construction-Site Images Using 3D CAD-Based Filtering. *Journal of Computing in Civil Engineering 24*(1), 56–64.

Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017, jul). Aggregated residual transformations for deep neural networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Volume 2017-Janua, pp. 5987–5995. IEEE.

Zagoruyko, S., A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár (2016). A MultiPath Network for Object Detection. *Proceedings of the British Machine Vision Conference*.

Zeiler, M. D. and R. Fergus (2014). Visualizing and Understanding Convolutional Networks. *Computer Vision-ECCV 2014 8689*, 818–833.

Zhu, Z. and I. Brilakis (2010, nov). Parameter optimization for automated concrete detection in image data. *Automation in Construction 19*(7), 944–953.

Declaration

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. The thesis was not previously presented to another examination board and has not yet been published.

Munich, September 3, 2018

Bernhard Mueller

Bernhard Mueller
bernhard.mueller@tum.de