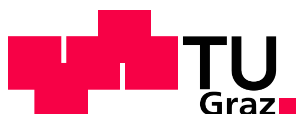


Christian Ertler, BSc

Deep Learning for Pedestrian Detection in RGB-D Images from an Elevated Viewpoint

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme

Computer Science

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Horst Bischof

Institute for Computer Graphics and Vision

Advisors

Dipl. Ing. Horst Possegger

Dipl. Ing. Michael Opitz

Graz, Austria, Oct. 2016

Abstract

Pedestrian detection is a crucial pre-processing step for many applications in the area of computer vision, including video surveillance, traffic control, or self-driving cars. We investigate a special setting of pedestrian detection in images taken from an elevated viewpoint capturing overhead views of people as part of an automated traffic control system. Compared to standard pedestrian detection tasks, such as autonomous driving, this special viewpoint causes significantly more pose-variation of pedestrians in images, as the appearance of pedestrians strongly depends on the relative position to the camera. Common assumptions about the pose of pedestrians are violated in this setting.

We employ state-of-the-art deep convolutional neural network (CNN) architectures designed for generic object detection tasks and fine-tune them on custom datasets, which we recorded and labeled for this thesis. Besides RGB images, these datasets also comprise depth images computed from stereo image pairs. We modify the network architectures to fuse both modalities inside the models. Further, we combine the detection architecture with another network to replace the standard greedy non-maximum suppression (NMS) algorithm. Our final model is able to perform pedestrian detection in RGB-D images without the need of any post-processing.

We show that CNNs, which have originally been trained on RGB data, benefit from an additional depth modality fused inside the network. Our best fusion model achieves $\approx 9\,\%$ relative improvement over the baseline RGB network, while inference takes about 87 ms on a modern middle class GPU. Additionally, we show the benefits of replacing hand-crafted NMS algorithms in object detectors by a trainable alternative. Our model improves over greedy NMS, especially in crowded scenes ($\approx 5.5\,\%$ relative improvement), imposing a negligible additional runtime cost of 1.6 ms during inference.

Kurzfassung

Personenerkennung ist ein essentieller Schritt in vielen Computer Vision Anwendungen. Beispiele dafür sind Videoüberwachung, Verkehrsüberwachung oder selbstfahrende Autos. In dieser Arbeit untersuchen wir ein spezielles Szenario der Personenerkennung in Bildern, die von einem erhöhten Blickwinkel aus aufgenommen wurden. Dieser Blickwinkel stellt bildbasierte Systeme vor zusätzliche Herausforderungen, da das Erscheinungsbild von Personen stark von der relativen Position zur Kamera abhängt. Dadurch müssen herkömmliche Annahmen über Posen von Personen verworfen werden.

Wir adaptieren moderne Netzwerkarchitekturen, die für generische Objekterkennung konzipiert wurden, für Datensätze, die für diese Arbeit aufgenommen und annotiert wurden. Neben RGB Bildern enthalten diese Datensätze auch Tiefendaten. Wir modifizieren die Architekturen, sodass beide Modalitäten innerhalb der Netzwerke kombiniert werden. Zudem erweitern wir das Erkennungsnetzwerk mit einem zusätzlichen Netzwerk, das die herkömmliche NMS Methode ersetzt. Das endgültige Modell ist in der Lage, Personenerkennung ohne Nachbearbeitung durchzuführen.

Wir zeigen, dass Convolutional Neural Networks, die ursprünglich mit RGB Bildern trainiert wurden, von der zusätzlichen Tiefenmodalität profitieren. Mit unserem besten Modell erreichen wir eine relative Verbesserung von $\approx 9\,\%$ gegenüber dem RGB Modell, wobei die Erkennung pro Bild ca. 87 ms in Anspruch nimmt. Zusätzlich zeigen wir die Vorteile des optimierbaren NMS Modells gegenüber klassischen NMS Algorithmen. Unser Modell erreicht eine relative Verbesserung von $\approx 5,5\,\%$ in Situationen mit hoher Personendichte, wobei die zusätzliche Laufzeit nur 1,6 ms beträgt.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis.

______________________	______________________
Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

______________________	______________________	______________________
Ort	Datum	Unterschrift

Acknowledgments

First of all, I would like to express my deep gratitude to my advisors Horst Possegger and Michael Opitz for their outstanding support during my whole thesis. Their ideas, hints, and clear questions to all my answers made my life a lot easier and paved the road for the completion of this thesis

Special thanks go to my supervisor Prof. Horst Bischof for giving me the opportunity to conduct my Master's Thesis at the Institute for Computer Graphics and Vision and for his excellent support in administrative and technical questions.

Last but not least I want to thank my family and friends for supporting me during my entire studies and life.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Pedestrian Detection . . . 5
1.3 Detection Pipeline . . . 7
1.4 Outline . . . 7

2 Background and Related Work **9**
2.1 Notation and Definitions . . . 9
2.2 Machine Learning and Classifiers . . . 10
2.2.1 Probabilistic Perspective . . . 11
2.2.2 Linear Classifier . . . 12
2.2.3 Loss Function and Optimization . . . 13
2.3 Neural Networks . . . 14
2.3.1 Neurons . . . 15
2.3.2 Activation Functions . . . 16
2.3.3 BackProp Algorithm . . . 16
2.3.4 Optimization . . . 18
2.4 Convolutional Neural Networks . . . 20
2.4.1 Data Arrangement and Concepts . . . 20
2.4.2 Layers . . . 22
2.5 Deep Convolutional Neural Networks . . . 23
2.5.1 Architectures . . . 23
2.5.2 Feature Generalization . . . 25
2.5.3 Optimization Challenges and Countermeasures . . . 26
2.6 RGB-D Imaging . . . 29
2.6.1 Depth Estimation from Stereo Images . . . 29

2.6.2 Convolutional Neural Networks with RGB-D Images 30

3 Object Detection with Convolutional Neural Networks on RGB-D Images 33

3.1 From Classification to Object Detection 33
3.1.1 Region of Interest Pooling 35
3.1.2 Region Proposal Network 36
3.1.3 Combined Architecture 36
3.2 Incorporating Depth Data 37
3.2.1 Modality Fusion 38
3.2.2 Depth Encoding 41
3.3 Optimization 41
3.3.1 Multi-task Loss 41
3.3.2 Training Process 43

4 Non-Maximum Suppression with Convolutional Neural Networks 45

4.1 Input Preparation 46
4.1.1 Input Grid 46
4.1.2 Input Maps 47
4.2 Network Architecture 48
4.3 Optimization 50
4.3.1 Training Targets and Loss 50
4.3.2 Initialization and Solver 51
4.3.3 Data Augmentation 51

5 Experiments and Evaluation 53

5.1 Performance Metrics 54
5.1.1 Precision and Recall 54
5.1.2 Average Precision 55
5.1.3 Proposal Coverage 55
5.2 Datasets 56
5.3 Implementation Details 58
5.3.1 Frameworks and Code 58
5.3.2 Hardware Setup 59
5.3.3 Training and Testing Procedure 59
5.4 Depth Fusion for Pedestrian Detection 59
5.4.1 Quantitative Analysis 61
5.4.2 Qualitative Analysis 65
5.4.3 RPN with Depth 66
5.5 Learning Non-Maximum Suppression 66
5.5.1 Quantitative Analysis 67

5.5.2 Qualitative Analysis . . . 69
5.6 Runtime Performance . . . 70

6 Conclusion and Outlook **73**
6.1 Conclusion . . . 73
6.2 Outlook and Future Work . . . 74

A List of Acronyms **75**

Bibliography **77**

List of Figures

1.1 Pedestrian intent prediction . . . 2
1.2 Illustration of a surveillance viewpoint and an overhead viewpoint . . . 3
1.3 Example images from a surveillance viewpoint and an overhead viewpoint . 4

2.1 Decision boundaries . . . 13
2.2 Two-layer neural network . . . 15
2.3 Activation functions . . . 17
2.4 Neuron volume . . . 21
2.5 Convolutional neural network . . . 23
2.6 Learned convolution filters . . . 24
2.7 AlexNet . . . 24
2.8 Epipolar geometry and canonical stereo setup . . . 31
2.9 Comparison of depth encodings . . . 32

3.1 *Region of Interest* (RoI) pooling layer . . . 35
3.2 Illustration of a *Region Proposal Network* (RPN) . . . 37
3.3 Fast R-CNN architecture . . . 38
3.4 Late modality fusion . . . 39
3.5 Early modality fusion . . . 40
3.6 Smooth L_1 norm . . . 43

4.1 Illustration of a *Tyrolean network* (Tnet) score map . . . 48
4.2 Tnet with region features . . . 50

5.1 *Campus* dataset examples . . . 57
5.2 *Vienna* dataset examples . . . 58
5.3 Precision *vs.* recall curve on the *Campus* test set . . . 63

5.4 Precision *vs.* recall curve of early and late fusion models 64
5.5 Qualitative comparison of RGB and early max fusion model 65
5.6 Region proposal comparison RGB *vs.* early max fusion 66
5.7 Precision *vs.* recall curves of greedy *Non-Maximum Suppression* (NMS) models and Tnet 68
5.8 Qualitative comparison of greedy NMS and Tnet (*Campus* dataset) 69
5.9 Qualitative comparison of greedy NMS and Tnet (*Vienna* dataset) 70

List of Tables

2.1 Mathematical notations 10

5.1 Confusion matrix 54
5.2 Quantitative comparison on the *Vienna* test-set with *Height Above Ground* (HAG)-coloring 61
5.3 Quantitative comparison on the *Campus* test set with HAG-coloring 62
5.4 Quantitative comparison on the *Campus* test-set with HHA-encoding 63
5.5 Greedy NMS *vs.* Tnet 67

1 Introduction

Contents

1.1 Motivation 1
1.2 Pedestrian Detection 5
1.3 Detection Pipeline 7
1.4 Outline 7

In this chapter, we introduce the reader to the initial problems and goals of this thesis. Additionally, we provide an overview of our general approach and main contributions, and finally give an outline of the rest of this work.

1.1 Motivation

In this thesis, we address the problem of pedestrian detection from an elevated viewpoint. This is useful for automated traffic light control systems which optimize the light signals based on pedestrian intent prediction. An example of the output of such an intent prediction system is illustrated in Figure 1.1.

While the intent of motor vehicles can be captured automatically by current traffic control systems with sensors embedded in the traffic lane or cameras, pedestrians are still required to actively interact with the system by pressing a button, indicating that they want to cross the street. Unfortunately, this causes long waiting times and impatient pedestrians might cross the street prematurely, putting themselves and others into danger. Additionally, such a system can not distinguish between situations of pedestrians pushing the button multiple times and multiple pedestrians waiting for the cross signal. However, this information can be beneficial for a traffic light control system in order to optimize the

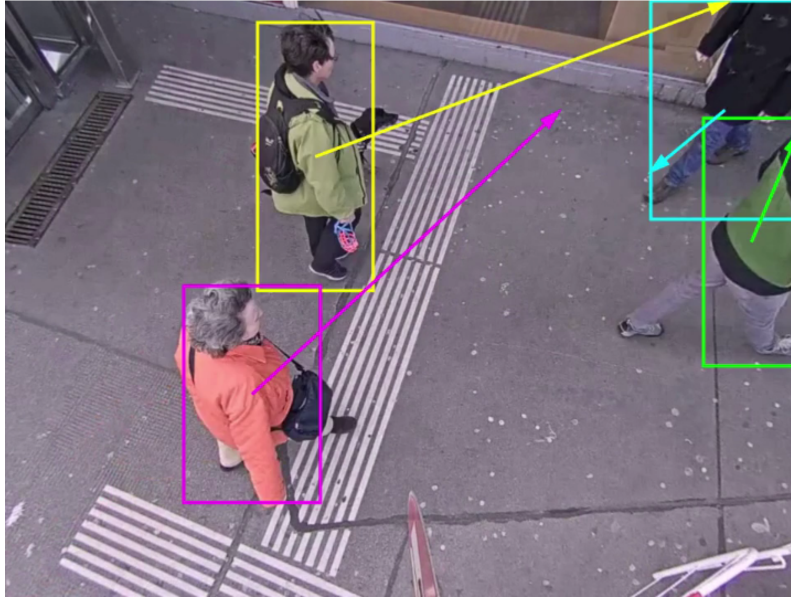


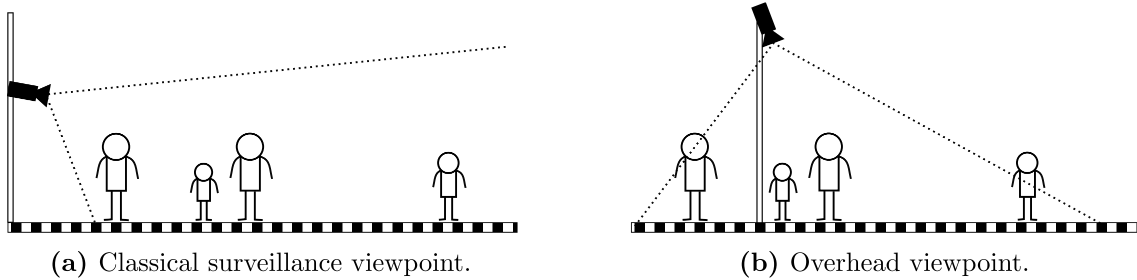
Figure 1.1: Illustration of pedestrian intent prediction. The coloured boxes correspond to pedestrian detections. The arrows indicate the intended destination estimated by the system based on the information of consecutive frames.

schedule of green phases, *e.g.*, by extending crossing times for larger crowds in order to allow them to safely clear the crosswalk.

To that end, we propose a stereo camera setup, which is mounted on top of the traffic light and records videos of pedestrians. In this thesis, we address the problem of pedestrian detection from this elevated viewpoint, which is an important step for predicting the intent of pedestrians, *i.e.*, whether they want to cross the road or not. The major research interests for pedestrian detection tasks in typical surveillance scenarios are side-view and front-view pedestrians. In these cases, the camera is mounted in a slightly elevated position in order to get a long-range field-of-view. In our traffic light control scenario, on the other hand, the cameras are highly elevated filming downwards in order to capture pedestrians close-by the crossroad. Figure 1.2 illustrates the differences between the two scenarios.

The overhead view in our scenario introduces significant differences in people's appearance and pose in images compared to those from a classical surveillance viewpoint. Figure 1.3 shows some example images from the two scenarios and illustrates some of these differences. In the following we briefly discuss these differences and the main challenges a vision-based system has to tackle in our scenario.

Appearance and pose Most pedestrian detection applications capture people from a side-view or only slightly elevated viewpoint. Thus, a common assumption is that



(a) Classical surveillance viewpoint.

(b) Overhead viewpoint.

Figure 1.2: Illustration of (a) a classical surveillance viewpoint with a long-range field-of-view and (b) an overhead viewpoint as used in our setting.

the general appearance of — at least unoccluded — people in such images, *i.e.*, their silhouette, is similar. Although intra-class variability and pose variation are common challenges in all pedestrian detection settings, the overhead viewpoint in our setup significantly violates this assumption.

The pose and appearance of a person strongly depends on the relative position to the overhead camera. Some parts of the person may be self-occluded. For example, the only visible parts of a person standing just beneath the camera are her head and shoulders. People at the border of the field-of-view, on the other hand, appear elongated and rotated about their vertical axis. Due to these transformations, we can not make assumptions about the location of certain parts inside the person's bounding box. By looking at the bottom row in Figure 1.3, we see that the head, for example, sometimes appears in the upper-left corner of the image (first example), in the upper-right corner (second example), in the lower-left (third example), or lower-right (fourth example) corner.

Since traditional single-template based approaches for pedestrian detection, such as *Histogram of oriented Gradient* (HoG) based detectors, can not cope with large pose variations, we utilize state-of-the-art deep learning based detectors. These detectors are capable of detecting objects with large pose variations, *e.g.* [22, 23, 55].

Bounding box ratios As appearance is affected by the viewpoint, so are the aspect-ratios of bounding boxes. The typical assumption of a nearly fixed aspect-ratio can not be applied in our case. This is also related to the variation in appearance and pose because rotation about the vertical axis of a person has a significant influence on the bounding box. Comparing the top and bottom row in Figure 1.3, we see that bounding boxes of people in classical scenarios are always upright, whereas the bounding boxes in our scenario range from upright, to quadratic, and up to

vertical. Note that those axis-aligned bounding boxes also contain significantly more background in our scenario compared to the surveillance scenario.

Clipped pedestrians Another problem introduced by the viewpoint is the increased number of pedestrians which are clipped to the image boundaries, and thus are not fully visible. Instead, only their legs or — depending on the position — only their torsos are visible. This is a consequence of the narrower field-of-view as illustrated in Figure 1.2. The question is how to detect these instances without harming the detection performance for fully visible people.

Figure 1.3: Example images of pedestrians from a classical surveillance viewpoint (top row) and an overhead viewpoint (bottom row). The images in the top row are taken from the KITTI dataset [21]. The images in the bottom row are from our custom datasets.

1.2 Pedestrian Detection

Object detection is one of the fundamental problems in computer vision and recognition. Given an image of a scene, the goal is to locate all object instances in that image and to assign a class label to them [18, 57]. In other words, the question is not only if an object of a specific class is present, but also where it is in the image. Therefore, object detection can be split in the following consecutive sub-tasks.

Classification This basic recognition problem is about assigning one or more class labels to images based on their content. The focus lies on the visual appearance of the objects in a scene. At this level, the image is regarded as a whole and the location of the contained objects is neglected. To that end, classification becomes even more challenging when the image contains multiple different objects or the object of interest is not the most salient one.

Localization Images are not constrained to contain only a single object of a unique class. Instead, many objects of the same (or even different) classes may be present. In this case, we are interested in where these objects appear in the image. The locations of the objects can be encoded by rectangles that tightly enclose the corresponding object instances, called bounding boxes or sometimes Regions of Interest (RoIs). To keep the task simple, bounding boxes are often constrained to be axis-aligned (*i.e.*, the rectangles are not rotated to better fit the object's boundaries). Nevertheless, the main objective is to find boxes that tightly fit the entire object and each object should be reported exactly once.

Computer vision poses the additional requirement to achieve these tasks by processing only image data without any further information. While humans are able to analyse a scene and effortlessly name and enumerate all of its objects, computers are still far away from reaching human performance in most scenarios. Especially when it comes down to the principal challenges all vision-based recognition problems have in common, the performance of computer vision algorithms can drop severely. Some of these challenges include significant variation in scale, illumination, viewpoint, and partially occluded objects. However, recent advances in the field of object detection and deep learning are promising to close this gap in the future.

Traditionally, object detectors are categorized into holistic HoG based detectors, *e.g.* [10], *Deformable Part Model* (DPM) based HoG detectors, *e.g.* [4, 13, 19], boosting based

detectors, *e.g.* [14, 72], and bag-of-words based detectors, *e.g.* [71]. HoG based and DPM based detectors operate in a sliding-window manner over the whole image, whereas bag-of-words based approaches rely on object proposals, in which features are extracted and classified in order to detect objects in images.

With the success of deep learning methods in recent years, detectors based on Convolutional Neural Networks (CNNs) were able to beat those methods on most object detection challenges. For example, OverFeat [59] uses a sliding-window regressor over the feature map produced by a classification CNN. In this way, it re-uses the representations of a classification model to compute bounding boxes for each class at every position of the feature map. Girshick *et al.*'s [23] R-CNN also makes use of a classification model by applying it to a set of pre-computed region proposals and feeding the resulting feature maps to class-specific Support Vector Machines (SVMs) to classify each region independently. Its successors Fast R-CNN [22] and Faster R-CNN [55] further improve over it by sharing computation on single images with RoI pooling and integrating the region proposal generation into the network. Other CNN based methods [45, 54] operate in a fully-convolutional way and do not rely on region proposals to generate detection hypothesises.

A special and well-studied case of object detection is *pedestrian detection*. As the name suggests, the objective is to detect pedestrians (or more generally people) in images. This *binary* object detection problem is often viewed as a standalone problem because of its importance to the broad field of *video surveillance* and its special challenges (*e.g.*, frequent inter-instance occlusions, heavy background clutter, and high intra-class variability).

To that end, several detectors and feature extractors such as HoG [10] and *integral channel features* [14] were primarily designed for pedestrian detection. Others adapted existing methods such as DPM for pedestrian detection [75]. While CNN based object detectors achieve a significant accuracy improvement on generic object detection, the improvement for pedestrian detectors is comparably small. Reasons for that include the high intra-class variability and the large range of scales in a single image. Li *et al.* [44] address the scale issue by extending Fast R-CNN [55] with built-in sub-networks operating at different scales. The outputs of the sub-networks are combined by a gate function defined over the sizes of the corresponding object proposals. Others are able to improve the performance by adding hand-crafted features [36].

Most previous work focuses on the problem of detecting pedestrians captured from a side-view. Our setting, however, is different since we detect pedestrians from an elevated viewpoint. This particular setup has received only little research interest. Ahmed and

Carter [2], for example, propose to project the image of a person to the center of the camera based on the known position before computing HoG features. However, they restrict their method to a top-view setup, where the ground plane is parallel to the camera. We do not make such strict assumptions about the camera's perspective. Furthermore, we process the image in a single feed-forward pass through a CNN, as opposed to each RoI separately.

1.3 Detection Pipeline

Typical object detection pipelines perform classification on smaller sub-windows of the image. This is either done in a sliding-window fashion or by pre-computing a set of region proposals that are likely to contain objects. Both approaches result in a dense set of detection scores along with many bounding boxes. Significantly overlapping boxes, however, likely belong to the same object instance. To prevent reporting multiple detections of the same object instance, *Non-Maximum Suppression* (NMS) is applied as a post-processing step.

In this thesis, we utilize recent advances in the area of deep learning to build an object detection pipeline that is trained and evaluated on our self-labeled datasets. We limit ourselves to CNN-based detectors, since their representational power, which is needed to learn the broad range of poses and appearances of pedestrians in overhead images, surpasses the one of other methods like DPM. Our main contributions are an extension of the state-of-the-art Faster R-CNN [55] method to RGB-D images fine-tuned for the task of pedestrian detection from an elevated viewpoint. Further, we combine it with a learnable alternative to NMS proposed by Hosang *et al.* [34]. The final detector detects pedestrians in images without any hand-crafted post-processing.

In order to be usable in real-world situations, we design our detector to meet the following properties:

- Reliability (detect most pedestrians and make few mistakes)
- Generalization (achieve good results in situations not present in the training data)
- Speed (detection should be possible at a high frame rate)

1.4 Outline

The remainder of this thesis is structured as follows. Chapter 2 starts with some background by evolving a simple linear classifier to deep convolutional neural networks. The theory

in this chapter forms the basis of the final models used for our detector. Chapter 2 also contains an extended discussion of related work.

The object detection model along with the classification and bounding box regression is described in Chapter 3. We also show different approaches to incorporate the depth data during training and inference.

Chapter 4 covers a trainable approach to NMS. We show how to transform the output of the detection model in order to train a network for NMS and how to replace the hand-crafted algorithm by this model. The combined network detects pedestrians in images without the need of a post-processing step.

Our experiments and evaluation results are described in Chapter 5. We perform both quantitative and qualitative analysis of the proposed models and compare our different approaches.

Finally, we conclude the thesis and discuss potential future directions in Chapter 6.

2 Background and Related Work

Contents

2.1 Notation and Definitions 9
2.2 Machine Learning and Classifiers 10
2.3 Neural Networks 14
2.4 Convolutional Neural Networks 20
2.5 Deep Convolutional Neural Networks 23
2.6 RGB-D Imaging 29

In this chapter we describe the theoretical background and related work needed for this thesis. We first introduce the mathematical notation used in this thesis. The main part of this chapter gives an overview of machine learning and convolutional neural networks in detail. The chapter concludes with a description of state-of-the-art deep learning models and theory about RGB-D imaging needed to compute depth images from our stereo setup.

2.1 Notation and Definitions

Before we start with the literature overview, we introduce the mathematical notations and definitions used throughout this thesis.

We use italic fonts (*e.g.*, y or y_i) to denote scalar values. Vectors are represented as bold lower-case letters and matrices as bold upper-case letters (*e.g.*, $\mathbf{x}$ or $\mathbf{W}$). Unless otherwise stated, all vectors are assumed to be column vectors. Subscripts are used to index vectors and matrices and the latter are ordered row-major (*e.g.*, $\mathbf{x}_i$ denotes the i-th element of vector $\mathbf{x}$ and $\mathbf{W}_{i,j}$ denotes the j-th element in the i-th row of matrix $\mathbf{W}$). The

Description	Notation
Scalar	y, c
Vector	$\mathbf{x}$
Matrix	$\mathbf{W}$
Indexing	$\mathbf{x}_i$, $\mathbf{W}_i$, $\mathbf{W}_{i,j}$
Transpose	$\mathbf{x}^\top$, $\mathbf{W}^\top$
Set	$\mathcal{P}$, $\mathcal{Y}$
Vector space	$\mathbb{R}^D$
Iteration value	$\mathbf{W}^{(i)}$
Mapping function	$f : \mathbb{R}^D \to \mathbb{R}^K$
Random variable	X, Y
Probability	$P\left(X = x\right)$
Conditional probability	$P\left(X = x \mid Y = y\right)$

Table 2.1: List of mathematical notations used throughout this thesis.

transpose of a vector or a matrix is written as $\mathbf{x}^\top$, or $\mathbf{W}^\top$ respectively. Sets are written in calligraphic letters (*e.g.*, $\mathcal{P}$) and vector spaces in double-lined upper-case letters with the superscript defining the dimensionality (*e.g.*, $\mathbb{R}^D$ or $\mathbb{R}^{K\times D}$). Values of a variable at a specific time or iteration in an algorithm are indicated by a braced superscript (*e.g.*, $\mathbf{W}^{(i)}$ is the matrix $\mathbf{W}$ at iteration i). Random variables are denoted by upper-case letters (*e.g.*, X, Y) and probability distributions over a random variable are written as $P\left(X\right)$. Probabilities of particular events are written as $P\left(X = x\right)$ — or in the case of conditional probabilities as $P\left(X = x \mid Y = y\right)$. An overview of the notations used in this thesis can be found in Table 2.1.

2.2 Machine Learning and Classifiers

In this thesis, we will use machine learning models to perform pedestrian detection in images. The two main approaches to machine learning are *supervised* and *unsupervised* learning [48]. While in the supervised approach we are given a set of labeled input-output pairs $\mathcal{D}_{\text{supervised}} = \{(\mathbf{x}, y) \mid \mathbf{x} \in \mathcal{X}, y \in \mathcal{Y}\}$, in the unsupervised approach we are only given the inputs $\mathcal{D}_{\text{unsupervised}} = \{\mathbf{x} \mid \mathbf{x} \in \mathcal{X}\}$. The training inputs $\mathbf{x}$ are multi-dimensional vectors, where each dimension is usually called a *feature*. If the output values y are categorial from a finite set, the machine learning problem is called *classification*. On the other hand, if they come from a real-valued set, the problem is called *regression*. We will address both, regression and classification problems in this thesis, though the emphasis lies on classification.

In machine learning, the entity responsible for learning and doing the actual classification on new instances is called a *classifier*. The purpose of a classifier is to find *decision boundaries* in the feature space $\mathcal{X}$ to divide it into disjoint regions. In a mathematical sense, a classifier can be seen as a mapping from the feature space to a discrete set of class labels. In the following, we will focus on parametrized classifiers. For a discussion on non-parametrized ones, we refer the interested reader to [48].

If we denote the set of possible parameters as $\mathcal{P}$ and the set of class labels as $\mathcal{Y}$, we can write the formal definition of the classifier's mapping $C : \mathcal{X} \times \mathcal{P} \rightarrow \mathcal{Y}$ as

$$\hat{y} = C(\mathbf{x}) = C(\mathbf{x}; \Theta), \tag{2.1}$$

with $\mathbf{x} \in \mathcal{X}$, $\Theta \in \mathcal{P}$, and $\hat{y} \in \mathcal{Y}$. The definition of a *regressor* is similar, beside that the output $\hat{y}$ will be continuous [48].

2.2.1 Probabilistic Perspective

Machine learning — and therefore classification as well — is often viewed from a probabilistic perspective. One can argue, that a classifier makes a probabilistic decision after observing samples drawn from probability distributions X and Y. The goal of a classifier is to evaluate the probabilities $P(Y = \text{"some class"} \mid X = x)$ *vs.* $P(Y = \text{"other class"} \mid X = x)$. The *Bayes' rule* gives us two different ways to achieve this:

$$\underbrace{P(Y = \text{"some class"} \mid X = x)}_{\text{posterior}} = \frac{\overbrace{P(X = x \mid Y = \text{"some class"})}^{\text{likelihood}}\overbrace{P(Y = \text{"some class"})}^{\text{prior}}}{\underbrace{P(X = x)}_{\text{evidence}}}. \tag{2.2}$$

A classifier that directly models the *posterior probability* is called a *discriminative* classifier, whereas a classifier that models the *likelihood probability* is called a *generative* classifier.

Intuitively, the difference between those two approaches is the view on the data: A discriminative classifier does not model the probability distribution of the data but only decides depending on the observed data. Therefore, it does not know how the data is distributed except for its observations. On the contrary, a generative classifier models the actual distribution of the data by both, the observations and the hidden information in the form of the prior. The classifiers used in this thesis are discriminative, since we are not interested in the general data distribution.

2.2.2 Linear Classifier

We now introduce a simple linear classifier which forms the basis of more powerful classifiers such as neural networks and convolutional neural networks. The latter are used in this thesis for the task of detecting pedestrians.

Given input samples $\mathbf{x} \in \mathbb{R}^D$, we can describe a simple linear mapping $f : \mathbb{R}^D \times \mathcal{P} \to \mathbb{R}^K$ to map the input sample vector to a vector of K class scores

$$f(\mathbf{x}) = f(\mathbf{x}; \Theta) = f(\mathbf{x}; (\mathbf{W}, \mathbf{b})) = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{2.3}$$

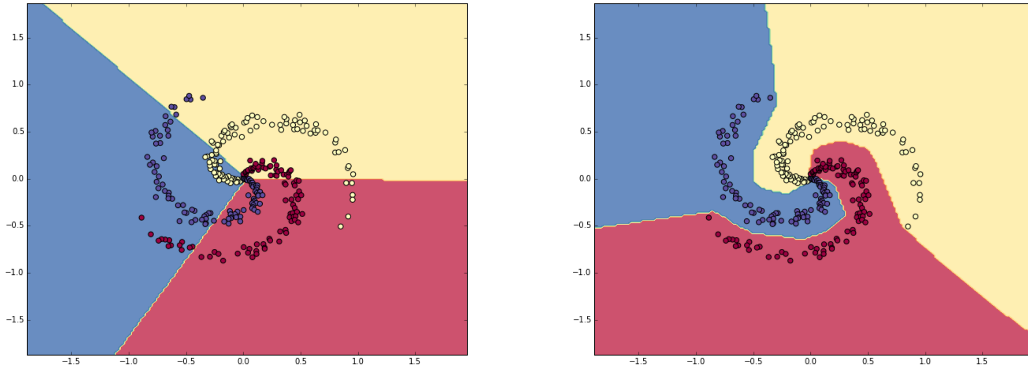
where $\mathbf{W} \in \mathbb{R}^{K \times D}$ is called the weight matrix and $\mathbf{b} \in \mathbb{R}^K$ the bias vector. Note that this is actually a combination of K separate classifiers, where each row $\mathbf{W}_k$ together with the bias $\mathbf{b}_k$ corresponds to a single classifier for class k. To be more precise, each pair specifies a hyperplane defining the zero-score border in the D dimensional feature space, with $\mathbf{W}_k$ being the normal vector of the hyperplane and $\mathbf{b}_k$ the translation factor of the hyperplane along $\mathbf{W}_k$.

To simplify the notation, we can apply the so-called *bias trick* [6] which allows us to combine the two parameters $\mathbf{W}$ and $\mathbf{b}$ by appending $\mathbf{b}$ as additional column to $\mathbf{W}$ and extending each sample vector $\mathbf{x}$ with one additional dimension holding the constant 1. Note the changes in dimensions with $\mathbf{x} \in \mathbb{R}^{D+1}$ and $\mathbf{W} \in \mathbb{R}^{K \times D+1}$ and the simplified definition of $f : \mathbb{R}^{D+1} \times \mathcal{P} \to \mathbb{R}^K$:

$$f(\mathbf{x}) = f(\mathbf{x}; \Theta) = f(\mathbf{x}; \mathbf{W}) = \mathbf{W}\mathbf{x}. \tag{2.4}$$

The linear combination of the model parameters $\mathbf{W}$ and the sample $\mathbf{x}$ gives the distances of the sample to the hyperplanes. Those distances can be interpreted as the scores of the classifier for each class. Figure 2.1a shows an example of the classification boundaries of such a linear classifier. Given suitable model parameters $\mathbf{W}$, the classifier is able to classify input samples which are linearly separable.

A simple way to allow a linear classifier to model data which are not linearly separable is called *basis function expansion* [6, 48]. By replacing $\mathbf{x}$ with a non-linear function $\phi(\mathbf{x})$, the classifier is able to classify input samples which are not linearly separable in the feature space but in the transformed feature space spanned by the basis functions $\phi(\mathbf{x})$. However, in Section 2.3, we describe a more powerful approach with adaptive basis functions that are learned by the model.



(a) Decision boundaries of a linear classifier. **(b)** Decision boundaries of a neural network.

Figure 2.1: Decision boundaries of different classifiers for a two-dimensional spiral point cloud. Note that this dataset is clearly not linearly separable. Images taken from [39].

2.2.3 Loss Function and Optimization

The goal of machine learning is to derive a model only from a given training dataset, *i.e.*, to learn the parameters of the model by providing a set of feature samples and the desired outputs of the model (*i.e.*, the labels). Before we extend the linear classifier to neural networks in Section 2.3, we describe loss functions and optimization, which is the process of finding good parameters.

For now, we have defined the *score function* f of the classifier in Equation (2.4). In order to optimize the model parameters, we need to define another function $L : \mathbb{R}^{K \times D+1} \to \mathbb{R}$ which is able to measure the performance of the model given the corresponding training labels $y \in \{1 \dots K\}$. The output of this so-called *loss function* should be low if the model fits the data well, and should be high otherwise.

There are many ways to define such a function. In general, the choice of the right loss function depends on the task we want to solve. A popular loss function for the task of multi-class classification is the *softmax cross-entropy loss*:

$$L(\mathbf{W}) = \frac{1}{N} \sum_{i}^{N} - \log \left(\frac{e^{f(\mathbf{x}_i;\mathbf{W})_{y_i}}}{\sum_j^K e^{f(\mathbf{x}_i;\mathbf{W})_j}} \right) \tag{2.5}$$

$$= \frac{1}{N} \sum_{i}^{N} -f(\mathbf{x}_i;\mathbf{W})_{y_i} + \log \sum_{j}^{K} e^{f(\mathbf{x}_i;\mathbf{W})_j}. \tag{2.6}$$

The fraction part in Equation (2.5) is the so-called *softmax* function that transforms a real-valued score vector to a vector of values in $[0, 1]$ that sum to 1. The resulting vector can be interpreted as a probability distribution. From Equation (2.6) we can see that this loss function actually computes the cross-entropy $H(p, q) = -\sum_x p(x)\ \log q(x)$ between the estimated class probability distribution and the distribution given by the class labels y_i. Intuitively, the softmax cross-entropy loss encourages the classifier to learn the "true" class probability distribution as approximated by the labels of the training samples.

Having the score function from Section 2.2.2 and the loss function L defined above, we can formulate the training of our classifier as a *minimization problem* to find the set of parameters $\mathbf{W}$ that minimizes the loss of the model. More formally, we seek the optimal weights $\mathbf{W}^*$ as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} L(\mathbf{W}). \tag{2.7}$$

In general, loss functions of linear classifiers are *convex*. In contrast to *non-convex* functions (*e.g.*, the loss function of a neural network), such functions only have global minima. Hence, *convex* optimization methods are typically used to optimize the weights of linear classifiers. In Section 2.3.4, we describe optimization methods which are used for state-of-the-art neural networks and CNNs.

2.3 Neural Networks

In this section, we extend the linear classifier from Section 2.2.2 to Neural Networks (NNs) by stacking multiple linear classifiers together and introducing *non-linearities* between them. We also describe *Back Propagation* (BackProp), which is an algorithm that helps to compute the gradient of a loss function w.r.t. all parameters in the network.

The score function f of a two-layer network is defined as

$$f(\mathbf{x}) = f\left(\mathbf{x}; \left(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\right)\right) = \mathbf{W}^{(2)} h\left(\mathbf{W}^{(1)}\mathbf{x}\right), \tag{2.8}$$

where $h(\cdot)$ is an element-wise non-linear *activation function*. Note the importance of the non-linearity: If we remove it, the two matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ can be written as a single matrix $\mathbf{W}$, resulting in the definition of the linear classifier in Equation (2.4). Figure 2.1b shows the classification boundaries of a neural network. The non-linearities allow the network to learn arbitrary decision boundaries.

2.3.1 Neurons

As already mentioned before, the notation in Equation (2.4) actually describes many classifiers, where each row of $\mathbf{W}$ corresponds to one classifier. We now define a *neuron* on layer j operating on M neurons:

$$\mathbf{z}_m^{(j)} = h\left(\mathbf{a}_m^{(j)}\right) = h\left(\sum_i^N \mathbf{W}_{m,i}^{(j)} \mathbf{z}_i^{(j-1)}\right), \tag{2.9}$$

where $\mathbf{z}^{(0)} = \mathbf{a}^{(0)} = \mathbf{x}$.

A feed-forward neural network or *Multi-layer Perceptron* (MLP) is a combination of multiple neurons arranged in sequential layers, where the input of each neuron in a layer is the output of all neurons in the previous layer. Hence, the network can be visualized as a directed, acyclic graph like in Figure 2.2. The leftmost layer is the so-called *input layer*, where each neuron outputs one dimension of the input sample. Note that the non-linearity is often omitted at the input neurons since they are just responsible to feed data into the network. The rightmost layer is called *output layer* and computes the final output of the network. The layer in-between is a *hidden layer* with the non-linearity applied to it. A neural network is not restricted to a single hidden layer. As we will see later in this thesis, modern neural networks consist of many hidden layers.

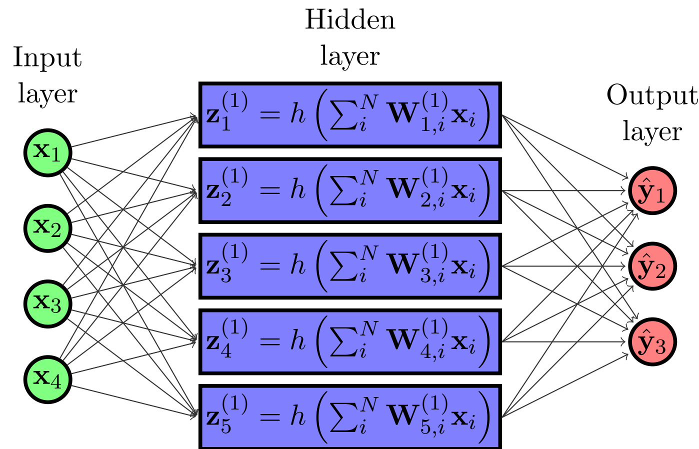


Figure 2.2: Illustration of a two-layer neural network with 4 input neurons, one hidden layer with 5 neurons, and 3 output neurons. The edges between the neurons correspond to a multiplication with the weights in $\mathbf{W}^{(1)}$ (green to blue) and $\mathbf{W}^{(2)}$ (blue to red).

2.3.2 Activation Functions

A neural network is in fact a combination of many inter-connected linear classifiers extended by a non-linearity. This non-linearity is the crucial difference to a linear classifier. It enables the network to classify samples based on many non-linear transformations. In contrast to the classifier in Section 2.2.2, which is only able to model linearly separable data, the decision boundaries of a neural network are not necessarily linear. In fact, it can be shown that a neural network — even with a single hidden layer — is a *universal approximator* which can model any suitably smooth function [48].

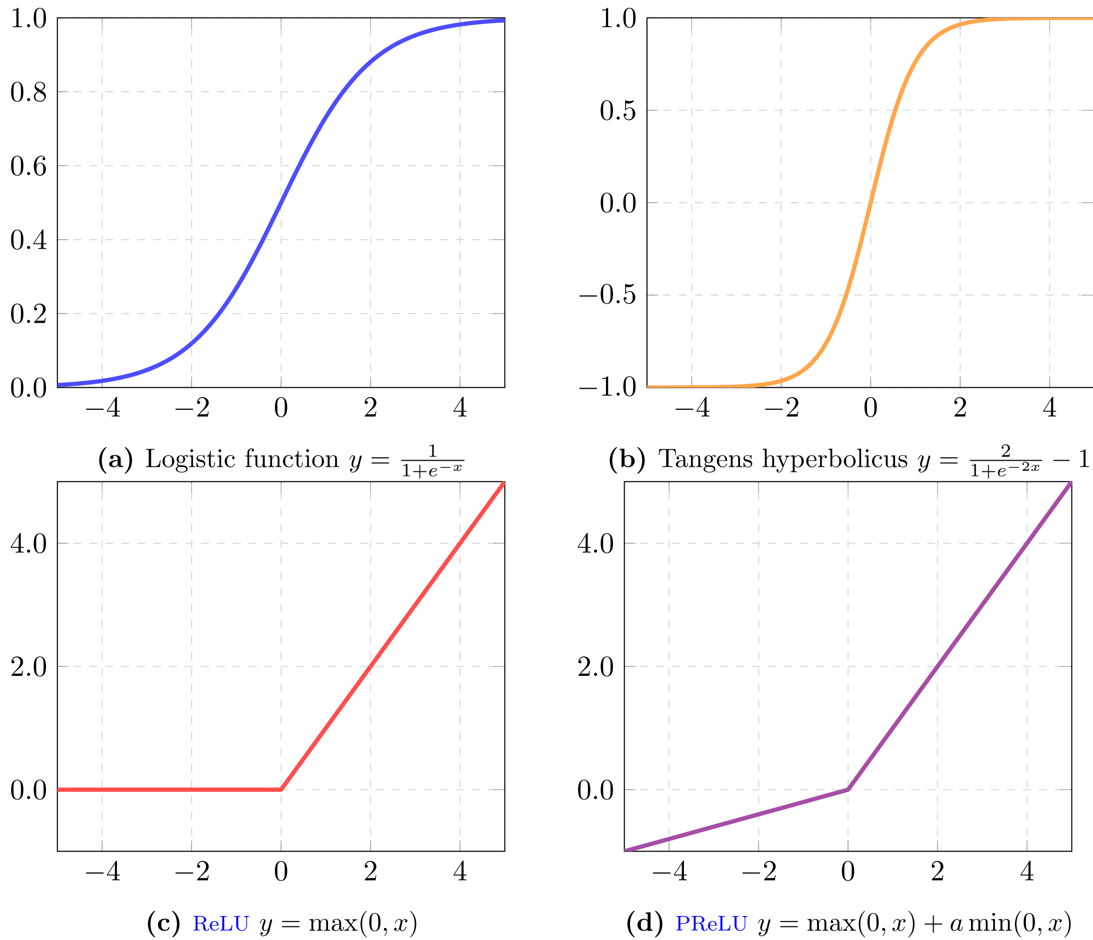
The non-linear functions are often called *activation functions* because they take the activations $\mathbf{a}_i^{(j)}$ as inputs and decide how much the network responds to them (*i.e.*, how much of the activation will be fed to the next layer's neurons). Historically, the most commonly used activation function was the *logistic* function, however, there are other suitable functions. The only requirement is the differentiability w.r.t. the parameters. Some commonly used activation functions are the *logistic* (or *sigmoid*) *function*, the *tangens hyperbolicus* and the *Rectified Linear Unit* (ReLU) [41, 49] as well as its parametrized form *Parametric Rectified Linear Unit* (PReLU) [31]. These functions are illustrated in Figure 2.3.

2.3.3 BackProp Algorithm

Training of neural netowrks is done in an iterative process which relies on evaluating the gradient of the loss function w.r.t. the parameters of the network (*i.e.*, the weights and biases). We now describe a framework called BackProp [6] to efficiently evaluate this gradient for a general feed-forward network with arbitrary structure and activation functions.

The core of BackProp is the *chain rule*, which makes it possible to compute the gradient stepwise backwards from the loss to the input layer. In general, BackProp allows to split the computation of the gradient of expressions involving multiple composed functions — such as the score function f of a neural network — into smaller parts, which are easier to differentiate. For example, if we see the computation of one neuron as denoted in Equation (2.9), the chain rule gives

$$\frac{\partial \mathbf{z}_m^{(j)}}{\partial \mathbf{W}_m^{(j)}} = \frac{\partial \mathbf{z}_m^{(j)}}{\partial \mathbf{a}_m^{(j)}} \frac{\partial \mathbf{a}_m^{(j)}}{\partial \mathbf{W}_m^{(j)}}, \tag{2.10}$$



(a) Logistic function $y = \frac{1}{1+e^{-x}}$ **(b)** Tangens hyperbolicus $y = \frac{2}{1+e^{-2x}} - 1$

(c) ReLU $y = \max(0, x)$ **(d)** PReLU $y = \max(0, x) + a \min(0, x)$

Figure 2.3: Plots of commonly used activation functions of neural networks.

thus, the computation of each gradient is a local process involving only the inputs and outputs of arbitrary sub-expressions. Due to this intermediate computations, BackProp can evaluate the gradients of arbitrary complex functions efficiently.

Another interpretation of BackProp is to see the function as a real-valued circuit or graph in which messages are passed forward (*i.e.*, data from the input layer to the output layer) and backwards (*i.e.*, gradients from the output layer to the input layer). Each node is a modular computation unit which is not only able to compute the output of the function itself, but also the local gradient w.r.t. its input values. In recent years, this resulted in powerful programming frameworks working with such computational graphs, *e.g.*, [1, 5]. Those frameworks are able to automatically compute the gradients for arbitrarily complex composed functions.

2.3.4 Optimization

Equation (2.7) defines the optimization of a classifier as a minimization problem of the loss function. In contrast to the loss function in Section 2.2.3, the loss function of a neural network is *non-convex*. While the simplest way of solving such a problem is to randomly choose a number of different parameters $\mathbf{W}$ and keep the best performing one (*e.g.*, the one providing the smallest value of $L(\mathbf{W})$), we are interested in more efficient approaches. A simple, yet powerful idea is to start with a random set of parameters $\mathbf{W}$ and iteratively refine them in small steps.

We now describe two algorithms which are used in this thesis to perform the optimizations needed to train our models:

Gradient Descent The core idea of gradient descent is to compute the first-order gradients of the loss function (*i.e.*, $\nabla L(\mathbf{W})$) w.r.t. the current parameters $\mathbf{W}$ and make steps in the negative gradient direction. This algorithm is also called *steepest descent* [6] as it always chooses the direction of the steepest descent given by the negative gradient. The iterative algorithm is defined as

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla L(\mathbf{W}^{(\tau)}), \tag{2.11}$$

where the bracketed superscript (τ) refers to the current iteration and η is a parameter of the algorithm defining the size of the current update step. More intuitively, this *step size* tells the algorithm how far it should go in the computed direction. As we will see later in this thesis, choosing the right step size is a crucial part of the optimization process and often a challenging problem.

Computing this gradient involves computing the score function for all samples in the dataset. This, however, is computationally too expensive. To overcome these limitations, online versions of gradient descent such as *Stochastic Gradient Descent* (SGD) or *Mini-Batch Gradient Descent* have been proposed. With this modification, the loss is not computed for the whole dataset anymore. Instead, at each iteration, the algorithm samples a subset of the training set, called a *mini-batch*, and computes the loss only over this subset. The idea behind this is that the different samples in the training set are correlated, and so the loss over a big enough mini-batch is a good approximation of the loss over the entire set [43]. Another advantage of stochastic gradient descent is that it is less prone to getting stuck in local minima,

since stationary points in the whole dataset are in general not stationary points in each mini-batch [6].

Another modification of the original gradient descent algorithm is gradient descent with *momentum* [56]:

$$\mathbf{V}^{(\tau+1)} = \mu \mathbf{V}^{(\tau)} - \eta \nabla L(\mathbf{W}^{(\tau)}), \tag{2.12}$$

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} + \mathbf{V}^{(\tau+1)}, \tag{2.13}$$

where $\mathbf{V}$ is the so-called momentum term which is a running average of previous update steps. This helps to escape from local minima and valleys in the loss function's surface, and to smooth step directions when the gradient directions rapidly oscillate between steps [53].

Adam Another optimization algorithm used within this thesis is the *Adam* solver [40]. Just like stochastic gradient descent, Adam uses first-order gradients to compute the update direction. While SGD uses a fixed step size, Adam computes adaptive learning rates for different parameters from estimates of first and second moments of the gradient. Adam combines the advantages of AdaGrad [16], which works well with sparse gradients, and RMSProp [69], which works well in online and non-stationary settings. The update step of Adam can be seen in the following equation:

$$\mathbf{M}^{(\tau)} = \beta_1 \mathbf{M}^{(\tau-1)} + (1-\beta_1)\nabla L(\mathbf{W}^{(\tau)}), \tag{2.14}$$

$$\mathbf{V}^{(\tau)} = \beta_2 \mathbf{V}^{(\tau-1)} + (1-\beta_2)\nabla L(\mathbf{W}^{(\tau)})^2, \tag{2.15}$$

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \frac{\sqrt{1-\beta_2^\tau}}{1-\beta_1^\tau} \frac{\mathbf{M}^{(\tau)}}{\sqrt{\mathbf{V}^{(\tau)}}+\epsilon}. \tag{2.16}$$

Note that the superscripts of β_1 and β_2 in Equation (2.16) denote actual powers. The initial values $\mathbf{M}^{(0)}$ and $\mathbf{V}^{(0)}$ are set to 0.

Both SGD and Adam rely on the efficient computation of the gradients w.r.t. the parameters of the model. As described in Section 2.3.3, BackProp is used to compute these gradients. The definition of the overall optimization process of neural networks (taking the example of SGD) can be found in Algorithm 1.

Algorithm 1: BackProp algorithm with stochastic gradient descent

Data: Input samples $\mathbf{X}$
Result: Optimized weights $\mathbf{W}^*$
initialize all weights $\mathbf{W}^{(0)}$;
initialize $\tau = 0$;
while *not converged or termination criterion reached* **do**
  draw random sample $\mathbf{x}^{(\tau)} \in \mathbf{X}$;
  forward pass: compute $f\left(\mathbf{x}^{(\tau)}; \mathbf{W}^{(\tau)}\right)$;
  backward pass: compute $\nabla L\left(\mathbf{W}^{(\tau)}\right)$ via chain rule to get the local gradient for each weight;
  update parameters with SGD: $\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla L(\mathbf{W}^{(\tau)})$;
  increment τ;

2.4 Convolutional Neural Networks

In Section 2.3, we described neural networks that take arbitrary inputs in the form of vectors. In this thesis, we work with image data which are naturally encoded as a rectangular matrix of numbers — or in the case of RGB images as matrix of three-dimensional tensors. While it is possible to train and evaluate neural networks with image data (*e.g.*, by vectorizing the matrix by horizontally stacking the transposed rows), recall that the neurons of each layer are fully-connected to the neurons of the neighboring layers. This results in a tremendous amount of parameters.

Considering reasonably sized input images of dimensions $500 \times 500 \times 3$, a single neuron in the first hidden layer has 750000 weights. Further, for image data, pixels which are far apart (*e.g.*, upper right pixel and lower left pixel) are rarely correlated with each other [6]. Thus, the combination of these pixels is not helpful for classifying objects in an image.

Moreover, this data arrangement completely ignores the complex two-dimensional spatial structure of an image [65]. We now describe CNNs. A CNN is a special form of a neural network primarily designed to work with image data.

2.4.1 Data Arrangement and Concepts

In order to represent image data in a more reasonable way, neurons of convolutional neural networks are usually arranged in three dimensions: (1) width, (2) height, and (3) depth, where depth denotes the third dimension of the neuron volume in this case. In the input layer, the depth dimension is usually of size three — one value for every color channel. As we will see later, the size of the third dimension usually grows in subsequent layers in order

to increase the expressiveness of the network. Thus, the output of each layer can be seen as a matrix of feature tensors often referred to as feature map. This data arrangement allows the network to take advantage of highly correlated features in local subregions of the image and to re-use learned weights for different spatial locations. Figure 2.4 shows an illustration of a convolutional neural network and its neuron arrangement.

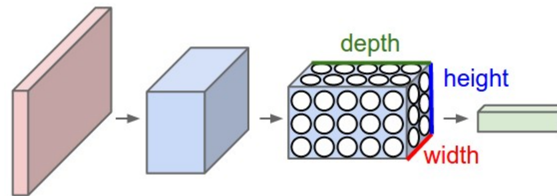


Figure 2.4: Illustration of a convolutional neural network and its neuron volume. Image taken from [39].

This data arrangement encourages the three core concepts of CNNs, namely (1) *local receptive fields*, (2) *weight sharing*, and (3) *subsampling* [6]:

(1) **Local receptive fields** Neighboring features (or pixels in case of the input layer) share more information than features appearing far away from each other. Hence, neurons should only be connected to neurons which are spatially near. The region including the connected neurons from the previous layer is called the local receptive field of a neuron. This concept — in combination with weight sharing — introduces an invariance to translations and distortions of the input image.

(2) **Weight sharing** Structures in images are translation invariant. For example, every natural image contains edges and we expect the network to learn filters reacting to those edges.

While a traditional neural network needs to learn neurons firing for specific edges for every possible location of that edge, neurons of a convolutional neural network share the learned weights for that edge with every neuron in the same depth channel of the three-dimensional neuron volume. Intuitively, one neuron learns a filter and applies it in a convolutional way at every spatial location in the input feature map, producing a new feature tensor for every location. Sharing weights in this way significantly reduces the number of parameters without loosing too much expressiveness of the network.

(3) **Subsampling** The last concept is called subsampling and introduces robustness to the exact spatial location of features. Subsampling can be achieved by dividing

the feature map into contiguous, (potentially) overlapping regions and squashing the feature tensors in these regions to a single tensor. Choosing subsampling regions of size 2×2 results in an output feature map of half the size of the input feature map.

2.4.2 Layers

Traditional neural networks consist of an input layer, one or more hidden layers, and an output layer, where each layer operates in a fully-connected way. Hence, to implement the three concepts described above, we need new types of layers which we describe now.

As it turns out, the concepts of receptive fields and weight sharing can be perfectly implemented by a discrete two-dimensional convolution operation performed over the feature map of the previous layer. The size of the receptive field is defined by the size of the convolution filter, which in turn is trainable via the BackProp algorithm. Each *convolutional layer* applies multiple different filters to the input feature map producing a three-dimensional output feature map again.

The concept of subsampling is implemented by means of so called *pooling layers*. Like a convolutional layer, a pooling layer performs a windowed operation on the input feature map. However, in contrast to a convolution, which is usually performed densely for every pixel, the window operation of a pooling layer is usually applied with a bigger stride, meaning that the operation is not performed for every pixel. Moreover, a pooling layer does not need any weights since it performs a static operation. The most commonly used type of pooling is *max-pooling* [41, 63, 67, 77], though other types of pooling operations, such as average pooling or stochastic pooling are possible. However, recent publications propose to replace pooling layers by other approaches, such as enlarging the *stride* (*i.e.*, the interval at which to apply the filters to the input feature map) of a convolutional layer [64]. A convolutional layer with a stride bigger than one performs subsampling as well.

While these new types of layers are at the core of convolutional neural network architectures, the layers of traditional neural networks are still important: Usually each convolutional and pooling layer is still followed by a non-linearity and many state-of-the-art networks conclude with one or more fully-connected layers. The optimization process does not change either, since both convolutions and pooling operations are differentiable. Thus, the BackProp algorithm from Algorithm 1 can be applied.

An illustration of a convolutional neural network architecture including convolutional, pooling and fully-connected layers can be seen in Figure 2.5. Note that the non-linearities are not explicitly drawn in this figure.

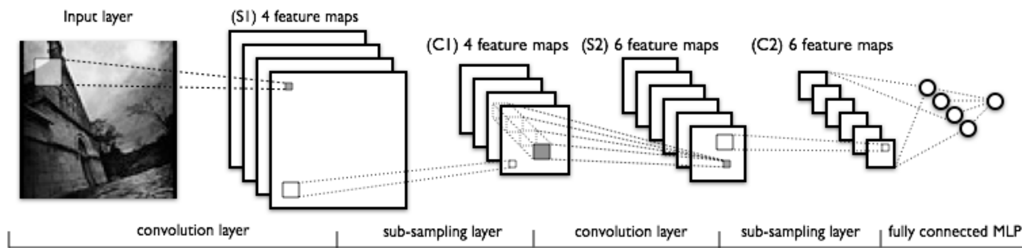


Figure 2.5: Illustration of a convolutional neural network with two convolutional and pooling layers followed by a fully-connected layer. Image taken from [11].

2.5 Deep Convolutional Neural Networks

Early convolutional neural networks showed very good performance on simple tasks like digit detection on the *MNIST handwritten digit database* [42]. However, those networks only consist of 2 convolutional and pooling layers and do not scale well to larger images. While those models are powerful enough for the tasks they are designed for, the representational power is restricted. Modern computer vision datasets — like the *ImageNet* database [12] and the *MIT Places* dataset [78] — consist of millions of images categorized in a huge amount of categories. Those datasets made it necessary to train significantly more complex models. Due to recent advances in *Graphics Processing Unit* (GPU) hardware, it is now possible to train such complex models which achieve near human level accuracy on large annotated image datasets [41]. We now describe *deep convolutional neural networks*, which are trained on large-scale image databases and are powerful enough to achieve state-of-the-art results on those datasets.

2.5.1 Architectures

As the word "deep" already suggests, the core idea of deep convolutional neural networks is to deepen the network by stacking multiple layers of convolutions and pooling operations in succession. The resulting models are actually inspired by the human visual cortex described by Hubel and Wiesel [37], which is thought to work with a hierarchy of simple and complex cells, where simple cells primarily respond to oriented edges and complex cells to spatial invariant combinations of them [65].

Figure 2.6 shows a visualization of the filter weights learned by a deep neural network on the ImageNet database. Just like in the visual cortex, the earlier layers learn *Gabor*-like filters that respond to generic low-level structures such as edges and corners, whereas layers

deeper in the network learn filters responding to more complex structures or even parts of actual objects [65].

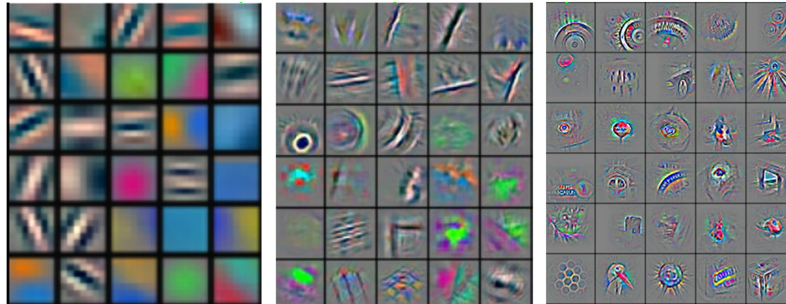


Figure 2.6: Visualization of learned filters of convolutional layers at different depths. Left: first layer. Middle: intermediate layer. Right: last layer. Image taken from [77].

One seminal work about deep neural networks is the work of Krizhevsky *et al.* [41], where they improved the performance on ImageNet object classification by a large margin. The network, called *AlexNet*, consists of five convolutional layers followed by three fully-connected layers. Each hidden layer is followed by a ReLU non-linearity. Max-pooling with overlapping windows is performed after the first, second, and fifth convolutional layer. Additionally, response-normalization layers follow the first and second convolutional layer. The filter sizes of the convolutional layers are 11×11, 5×5, and three times 3×3. Figure 2.7 shows a sketch of the network. We will use a slightly adopted version [77] of this network later in this thesis.

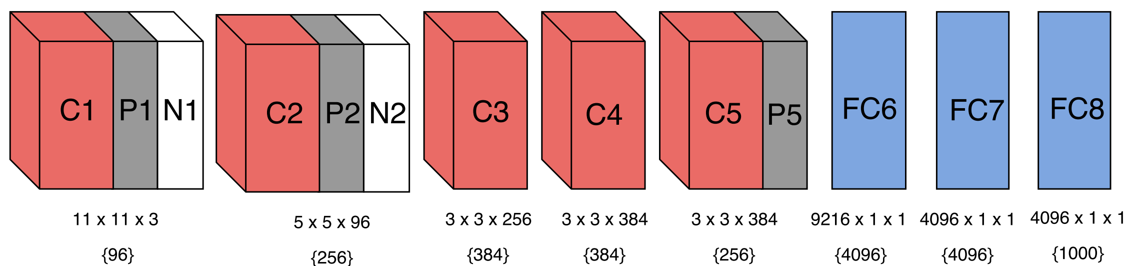


Figure 2.7: Illustration of the AlexNet model from Krizhevsky *et al.* [41]. The red boxes are convolutional layers, the grey ones are max-pooling layers, the white ones are local response normalization layers, and the blue ones are fully-connected layers. The numbers in the curly braces are the number of filters. The dimensions of the filters are represented by the numbers above them. Image taken from [65].

Subsequently published architectures are even deeper: The *VGG* model [63] consists of 13 convolution layers with small receptive fields of 3×3 followed by three fully-connected

layers. The authors show the importance of the depth of the network while keeping the number of parameters relatively small by using smaller convolution kernels.

The winner of the *ILSVR 2014* challenge was the so-called *GoogLeNet* [67]. The biggest contribution of this model is the *inception module*, which consists of different convolutions and pooling operations — all operating on the same input activation map. The concatenated activations of those layers form the final output activations of the inception module. In this way, the designer of the architecture does not have to make a choice for the right filter size or operation at each level of the network. Instead, the network can learn the right one or even a combination of all of them. Further, the inception module contains branches with and without max-pooling operations, which allows to extract multi-scale features.

He *et al.* [30] address the question "Is learning better networks as easy as stacking more layers?". Other work reported a fast saturation or even a degradation of test and training accuracy with very deep networks [28, 66]. This degradation problem indicates that solvers struggle to optimize deep networks, since in theory, deeper networks should perform at least as well as shallower networks by simply learning an identity mapping in the additional layers. *ResNet* [30] resolves this issue by adding *shortcut connections* in the form of element-wise additions of the input activations and the output activations of layers. By adding those connections, the layers are actually learning *residual mappings* instead of unreferenced mappings. In this way, they show that traditional solvers can optimize much deeper networks. Their winning submission to the *ILSVR 2015* challenge consists of 152 layers, thus it is almost 8 times deeper than the VGG network [63] while having a lower complexity in terms of the number of parameters.

2.5.2 Feature Generalization

One crucial advantage of deep neural networks trained on large-scale datasets is the possibility to re-use the models or the features produced by the models due to their *generalization* capabilities [65, 76]. Most of the state-of-the-art models are publicly available on the internet, and a lot of work — including this thesis — is based on those models.

In general, there are two ways how to re-use pre-trained deep neural networks for tasks which the networks were not intended in the first place:

(1) **Fine-tuning** Taking a model trained for a specific task (*e.g.*, trained for image classification on the ImageNet dataset) and adjusting it for a new task is called fine-tuning or transfer learning [76]. This can be achieved by simply taking the weights of a pre-trained model as initialization for further training on a new dataset

or even for a completely new task like object detection. The output layer of the network must be replaced by a new layer fulfilling the requirements of the task. Depending on the difference of the new task and the original one, one does not have to re-train all layers of the network. As already explained, earlier layers (and especially the first one) tend to learn very generic filters, while deeper layers can be very specific to the original data. Thus, if a model should be fine-tuned for a very similar task or data, the first few layers are often fixed, *i.e.*, their weights will not be updated during fine-tuning. On the other hand, if the new task is very different, it may be beneficial to fine-tune all layers. Another property worth considering is the size of the new dataset: Small datasets can lead to overfitting when fine-tuning all layers of a deep network [65].

Additionally, the hyperparameters of the training process must be re-evaluated. A good starting point is to choose a learning rate which is an order of magnitude lower than the one originally used [17, 23, 65].

(2) **Feature extraction** Before deep neural networks beat most of the previous state-of-the-art methods in many tasks, hand-crafted feature descriptors such as *Scale-Invariant Feature Transform* (SIFT) [46] or HoG [10] were very important. However, the features extracted from the last layers of deep convolutional neural networks are shown to be very descriptive too. In fact, it was shown that features extracted from deep neural networks can outperform (and therefore replace) hand-crafted features on many tasks in computer vision [15, 61].

Features can be extracted from a network by computing the forward pass of the network and saving the output features of the desired layer. Since the size of the last fully-connected layer is usually task specific, it is often omitted and the output of the last hidden layer is used as generic feature descriptor.

Feature extraction can be superior to fine-tuning a model if the new dataset only consists of very few samples, because fine-tuning on these samples may lead to dramatic overfitting [15].

2.5.3 Optimization Challenges and Countermeasures

In deep learning, data is represented in a high-dimensional space. Hence, the parameter space is high-dimensional as well. Additionally, in contrast to linear classifiers, more powerful classifiers such as deep neural networks have to optimize a *non-convex* loss

function to find the parameters of the model. Hence, there are many local optima and we want to find a global optimum or at least a "good enough" local optimum. To alleviate the problem of getting stuck in "bad" local optima, we use momentum (see Equation (2.12)).

Another closely related issue is that we want the classifier to model the "true" distribution of the underlying data, but in fact it only sees a small subset of the distribution given by the (limited) set of training samples. Thus, it is not guaranteed that the found optimum actually yields a good approximation of the "true" distribution. The goal is to find a model which generalizes well to unseen future data. If the model performs well on the training set, but poorly on unseen data, we say that the model *overfits* to the training set. We now describe some countermeasures to avoid these problems:

Regularization One reason for overfitting is that a complex model is fitted to a limited amount of training samples. Since the training samples typically contain noise and are only a sub-set of the "true" data distribution, a complex model starts to model the outliers of the data rather than the true underlying function. One way to overcome this issue is to artificially constrain the complex model and make it simpler. This is done by making the parameters small (*i.e.*, close to zero), which reduces the impact of individual parameters. To achieve this, the loss function is augmented with a regularization term to penalize large parameters. A common choice for the regularization term is the L_2 norm of the weights (which is also called the *Tikhonov regularization*). Thus, the loss function is extended as

$$L(\mathbf{W}) = L_{\text{data}}(\mathbf{W}) + \lambda ||\mathbf{W}||_2^2, \tag{2.17}$$

where L_{data} refers to the original definition of the loss in Equation (2.6) and λ is a parameter to adjust the influence of the regularization.

Dropout While deeper networks are suitable for more complex tasks, the larger amount of parameters also introduces a higher risk of overfitting. Traditional regularization approaches, such as L_2 regularization, do not suffice to train deep neural networks [65].

An additional technique used by many modern architectures is called *Dropout* [32]. A dropout layer randomly drops activations of input neurons with a probability of p during training. These random drops prevent co-adaption of neurons by only training a subset of all neurons of a layer in a single training batch. Since all neuron activations are used at inference time, they have to be scaled with factor $1/p$. One interpretation of dropout is that it efficiently trains an ensemble of networks with

shared weights [65]. One disadvantage of dropout is that it roughly doubles the number of iterations the solver needs to converge [41].

Other publications related to Dropout are DropConnect [73], Fast Dropout [74], and Maxout [24]. They have been shown to improve performance or reduce training time in certain circumstances, though they are not as frequently used as Dropout.

Monitoring the training process In order to have better control over the optimization, it is always a good idea to monitor the evolution of the loss during the training process [7]. Since the goal is to avoid overfitting to the training set, a subset of it is usually excluded from the training process. This subset is called the *validation set* and is useful to make a diagnosis of the optimization. If the loss on the validation set (or some other performance measure of interest) is starting to increase, the training can be stopped in order to avoid overfitting. This is usually called *early stopping* [6]. However, there are also other benefits. One can increase the step size of the solver when the validation loss plateaus in order to escape from a local minimum, *e.g.*, as done in [30, 31, 77].

Parameter initialization The choice of good initial values for the model parameters is crucial for many machine learning algorithms, especially for deep neural networks. While constant initialization prevents many models from learning meaningful patterns in the data, a good starting point is *random initialization*, where all initial weights are drawn from a Gaussian distribution. As the networks grow deeper, the initial weights become even more important for a successful training, since first-order optimization methods such as SGD can fail without good initialization. Krizhevsky *et al.* [41] are able to train their 8-layer AlexNet with carefully optimized random initializations for each layer.

For even deeper models (*e.g.*, ResNet with more than 100 layers), the choice of the initialization is very challenging. There are more sophisticated approaches to parameter initialization. Two recently published methods try to eliminate this task by introducing heuristics based on the network architecture. While both methods draw the initial weights from Gaussian distributions with zero mean, the parametrization differs. The *Xavier* initialization from Jia *et al.* [38] chooses the standard deviation as $\sqrt{\frac{1}{n_{\text{in}}}}$, whereas *MSRA* from He *et al.* [31] chooses a standard deviation of $\sqrt{\frac{2}{k^2 d_{\text{in}}}}$, where n_{in} is the number of input neurons, k the filter size of the layer, and d_{in} is the number of filters in the previous layer.

Input normalization Since we want the solver to update all parameters of the model in a reasonable amount, it is helpful to perform some kind of normalization of the input. A widely used technique is to normalize all inputs such that they have zero mean, their co-variances are approximately equal and all input variables should possibly be uncorrelated [43].

Data augmentation The training of large models usually requires large training sets. Unfortunately, recording and labeling of data is a very time consuming and error-prone task. Artificially generating new training data from already existing samples by applying small transformations to the data can improve generalization performance of neural networks. Examples of such transformations in the case of image data include rotating, translating, flipping or RGB jittering [41].

2.6 RGB-D Imaging

We will not only use traditional color images but also depth images in this thesis. We call images with three color channels (red, green, and blue) and one additional channel representing depth an *RGB-D image*. Note that we define the depth channel in two different ways depending on the origin of the image: (1) the *horizontal disparity*, which is the horizontal difference of the corresponding points of the left and right images in a stereo setup [50], or (2) as the normalized distance of the real-world object corresponding to the pixel in the image from the camera's projection center. Note that the disparity is inversely proportional to the distance and therefore sometimes called the *inverse depth* [68].

We now briefly describe the acquisition of RGB-D images from a stereo setup. Afterwards we elaborate different ways to represent this data in convolutional neural networks.

2.6.1 Depth Estimation from Stereo Images

A stereo setup consists of two cameras looking at the same scene from slightly different viewpoints. The rigid transformation between the two camera centers is known. In order to find the depth of a 3D point, we have to solve a *correspondence problem* [27], *i.e.*, finding pairs of points from the left and the right image plane corresponding to the same 3D point in the scene.

Finding pair-wise correspondences in images requires a significant computational effort, especially in the dense case where we want to have a depth value for every pixel in the image. Fortunately, the knowledge of the rigid transformation between the camera centers

can be exploited to reduce the number of correspondence candidates by means of the so-called *epipolar geometry* [27, 68]. As illustrated in Figure 2.8a, the intersection points of the ray connecting the two camera centers C_l and C_r with the image planes — or in other words the projection of one camera center into the other image plane — are the *epipoles* e_l and e_r. Given a real-world point U and its projected image point u_l in the left camera, we know that the corresponding point u_r in the right image plane must lie on the projection l_r of the epipolar line l_l (connecting u_l and e_l) into the right camera. The relation between a point in one image plane and the corresponding epipolar line in the other image image plane is given by the *fundamental matrix* $\mathbf{F}$, which can be computed linearly by knowing at least 8 point correspondences [27]. Once estimated, $\mathbf{F}$ is valid as long as the rigid transformation between the cameras does not change. In the calibrated case with the cameras' intrinsic parameters known, the *essential matrix* $\mathbf{E}$ is used.

Knowing $\mathbf{F}$ or $\mathbf{E}$, we can apply *image rectification* such that all epipolar lines are parallel to the horizontal axis of the image and thus, the search space for corresponding points can be reduced to a horizontal line. Geometrically, this rectification is equivalent to transforming the camera setup to the *canonical stereo configuration* (Figure 2.8b), such that the cameras are looking perpendicular to the baseline (*i.e.*, the line joining the camera centers C_l and C_r) and both "up vectors" look in the same directions [68], followed by re-scaling to compensate different focal lengths.

This transformation improves the performance and reliability of the correspondence search and leads to the simple inverse relationship of depth P_z and horizontal disparity d

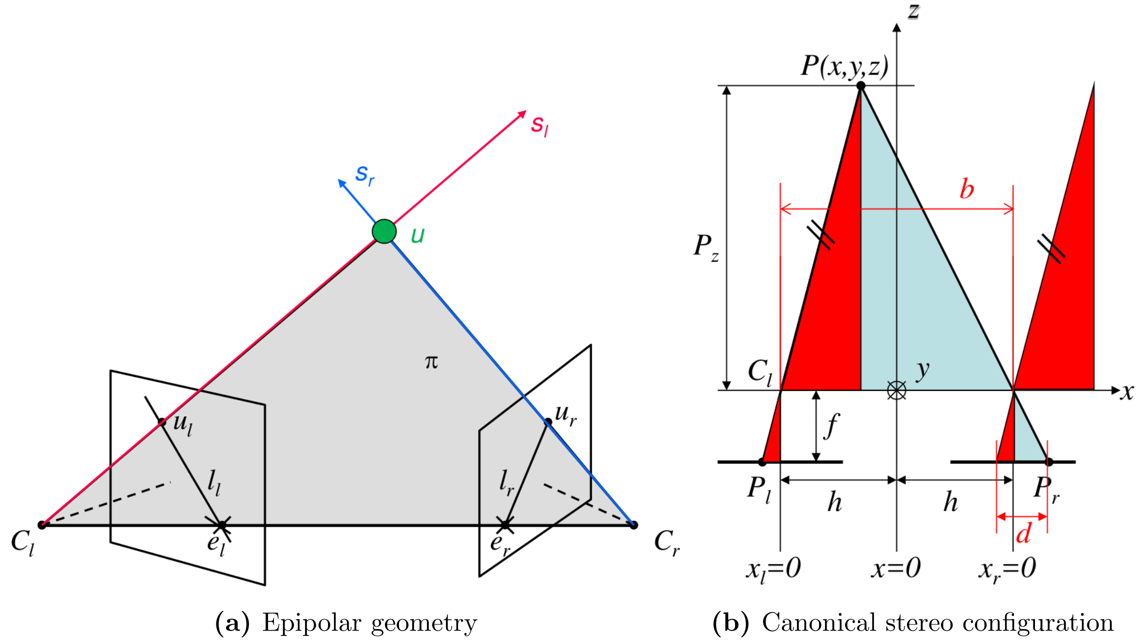
$$P_z = \frac{bf}{d}, \tag{2.18}$$

with $b = 2h$ being the distance of the camera centers and f the focal length of the cameras.

Computing disparity values densely for every pixel in an image is a more difficult problem, since finding the right correspondences in textureless regions is challenging due to the missing expressiveness of the pixels [68]. To overcome this issue, various local (window-based) and global re-fining algorithms like in [33] and [62] were introduced.

2.6.2 Convolutional Neural Networks with RGB-D Images

Thanks to their three-dimensional volume approach, convolutional neural networks are very flexible concerning the number of input channels. There is no design criteria forbidding to build networks which operate on four channels or more [3, 9, 58]. However, as already stated in Section 2.5.2, the power of modern deep convolutional neural networks lies in



(a) Epipolar geometry **(b)** Canonical stereo configuration

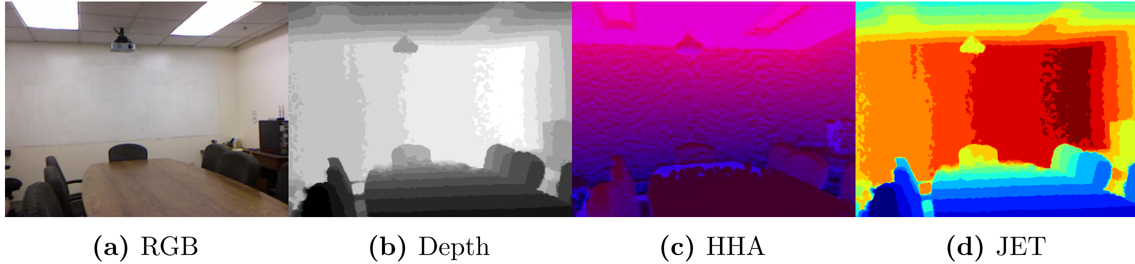
Figure 2.8: Illustration of (a) epipolar geometry and (b) the canonical stereo configuration. Images taken from [52].

feature re-using or fine-tuning of pre-trained networks. Those networks were designed for RGB images and rely on input images with three channels, though.

In [25], the authors show the possibility of fine-tuning a network trained on ImageNet RGB images with RGB-D images. They propose the *HHA encoding*, which represents each depth pixel by three features, namely (1) horizontal disparity, (2) height above ground, and (3) the angle between the pixel's surface normal with the gravity direction in the scene. Each channel is linearly scaled to map the values to the range $[0, 255]$. They argue that it is unlikely for a convolutional neural network to learn features encoding similarly expressive properties given limited training data. Further, they show experimentally, that HHA images share enough common structure with natural RGB images, such that they can be used to fine-tune a convolutional neural network which was originally trained with RGB images.

While the method proposed in [25] relies on two distinct networks — one for RGB, and one for HHA images — the network in [17] fuses the two streams of convolutional layers with fully-connected layers at the end of the network. Additionally, they replace the HHA encoding from [25] by a much simpler encoding, where they apply a *colormap* (*e.g.*, *jet*) to the depth image. In their experiments, the colormap encoding slightly outperforms the HHA encoding. The results suggest that the network is able to learn rich depth features

without manually engineered input features. Figure 2.9 shows a visual comparison of the different encodings.



(a) RGB **(b)** Depth **(c)** HHA **(d)** JET

Figure 2.9: Comparison of different depth encoding strategies. (a) The original RGB image. (b) Depth image with dark color corresponding to near, and bright colors to far pixels. (c) Pseudo-color visualization of the HHA encoding. (d) JET coloring. Note how the HHA encoding emphasizes discontinuities in the image in contrast to the JET encoding.

Both methods described above use the weights of a pre-trained RGB model as initialization for fine-tuning with depth images. Gupta *et al.* [26] propose a technique to transfer supervision between images from different modalities. They address the issue of having large-scale labeled datasets of RGB images but only a few small labeled datasets for depth images and a large amount of unlabeled RGB-D images. Their *supervision transfer* teaches a depth network to reproduce the mid-level representations of a well-trained RGB network by showing both networks unlabeled pairs of RGB and depth images during training, and minimizing a Euclidean loss of the two outputs. The gradient of the loss, however, is only back-propagated to the depth network. The resulting weights of the depth network serve as a better initialization for fine-tuning of the actual task.

3

Object Detection with Convolutional Neural Networks on RGB-D Images

Contents

3.1 From Classification to Object Detection 33
3.2 Incorporating Depth Data 37
3.3 Optimization 41

In this chapter, we will show how convolutional neural networks can be applied to the task of *object detection*. Early publications in this field proposed sliding window approaches [47, 60], where the network is evaluated for every sub-window of the image. Sermanet *et al.* [59] proposed a more powerful approach by evolving a CNN architecture stepwise from classification to localization and, finally, to detection. They make use of the inherent efficiency of CNNs when applied in a sliding window fashion by replacing the last fully-connected layers with 1×1 convolutions and thus, the network operates in fully-convolutional manner. However, at the time of writing, the state-of-the-art method for object detection with convolutional neural networks is called *R-CNN* [22, 23, 55]. In this thesis, the ideas of these methods are reused and extended for RGB-D data.

3.1 From Classification to Object Detection

With the release of the ImageNet database, convolutional neural networks became very popular for the task of image classification. Hence, a natural question is how to transfer this success to the task of object detection.

Girshick *et al.* [23] address this question with *R-CNN*. They apply a classification network to different sub-regions of an image, producing highly descriptive features for each of them. Instead of a sliding window approach, they use class-agnostic bottom-up region proposals generated by the *selective search* algorithm [70]. Finally, the extracted features are used to classify the content of the regions by a set of linear SVMs, each optimized for a single class.

Additionally, class-specific *bounding box regressors* are applied to the region proposals in order to get more precise locations: Each proposal box is represented by a tuple $P = (P_\mathrm{x}, P_\mathrm{y}, P_\mathrm{w}, P_\mathrm{h})$, where P_x and P_y are the pixel coordinates of the box's center and P_w, P_h are the width and height of the box. Similarly, the corresponding ground truth boxes are encoded as $G = (G_\mathrm{x}, G_\mathrm{y}, G_\mathrm{w}, G_\mathrm{h})$. The transformation applied by the regressor is defined as

$$\hat{G}_\mathrm{x} = P_\mathrm{w} d_\mathrm{x}(P) + P_\mathrm{x}, \tag{3.1}$$

$$\hat{G}_\mathrm{y} = P_\mathrm{h} d_\mathrm{y}(P) + P_\mathrm{y}, \tag{3.2}$$

$$\hat{G}_\mathrm{w} = P_\mathrm{w} e^{d_\mathrm{w}(P)}, \tag{3.3}$$

$$\hat{G}_\mathrm{h} = P_\mathrm{h} e^{d_\mathrm{h}(P)}. \tag{3.4}$$

While in [23], d_x, d_y, d_w, and d_h were modeled as independently optimized and parametrized linear functions of features extracted from the network, successive work [22, 55] proposed to infer the bounding box regression offsets directly from the network. The offsets are defined by Equations (3.5) to (3.8). Details about the training can be found in Section 3.3.

$$t_\mathrm{x} = \frac{G_\mathrm{x} - P_\mathrm{x}}{P_\mathrm{w}}, \tag{3.5}$$

$$t_\mathrm{y} = \frac{G_\mathrm{y} - P_y}{P_\mathrm{h}}, \tag{3.6}$$

$$t_\mathrm{w} = \log\left(\frac{G_\mathrm{w}}{P_\mathrm{w}}\right), \tag{3.7}$$

$$t_\mathrm{h} = \log\left(\frac{G_\mathrm{h}}{P_\mathrm{h}}\right). \tag{3.8}$$

The whole detection process of this system involves multiple sequential stages at inference and training time: (1) *region proposal*, (2) *region classification*, and (3) *bounding box refinement*. We will now describe various improvements to this system, which are re-used in this thesis in order to merge those stages to a single-stage algorithm.

3.1.1 Region of Interest Pooling

In terms of speed, the biggest drawback of R-CNN comes from the issue that, for every single proposal box, a forward pass through the network is necessary — even if the proposals come from the same image.

He *et al.* [29] introduced *spatial pyramid pooling networks* to speed up the process by sharing computation of CNN layers for boxes in the same image. The improvement comes from the *spatial pooling layer*, which is a drop-in replacement for traditional pooling layers. This layer is able to pool the features of arbitrarily sized feature maps to a fixed-sized output feature map.

Spatial pyramid pooling is further improved in [22] by means of the RoI pooling layer. It takes a list of proposal boxes as well as the shared feature map as input, and max-pools the features in each box to fixed-size output feature maps. Given an $h_{\text{in}} \times w_{\text{in}}$ proposal window (down-scaled to match the receptive field of the input feature map), the layer produces an $h_{\text{out}} \times w_{\text{out}}$ output feature map by dividing the input region into $\frac{h_{\text{in}}}{h_{\text{out}}} \times \frac{w_{\text{in}}}{w_{\text{out}}}$ sub-regions and computing the max values in each sub-region — just like a traditional max-pooling layer does. Note that the layer actually changes the batch-size to the number of boxes, since every box produces its own feature map. Figure 3.1 shows how the layer can be used to share the computation for many proposal boxes of the same image.

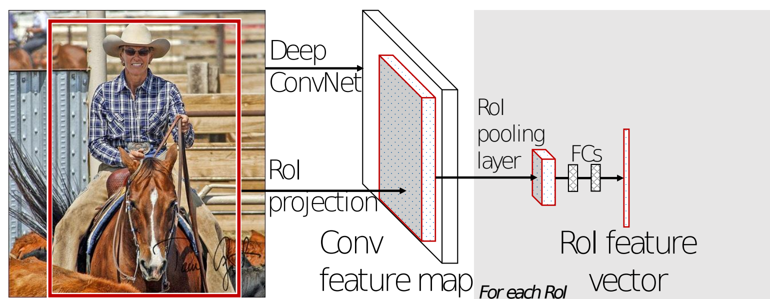


Figure 3.1: Illustration of a convolutional neural network utilizing the RoI pooling layer to share computation for proposal boxes. First, the image is passed through convolutional layers. Then, the RoI is projected onto the smaller feature map produced by the convolutional layers. The pooling layer pools fixed-size feature maps from the regions. Due to the fixed size of the output feature maps, it is possible to pass them through fully-connected layers – regardless of the actual size of the region. Image adapted from [22].

3.1.2 Region Proposal Network

The region of interest pooling layer is the first step towards a single-stage detection algorithm. However, there is still the task of proposal generation. The widely used proposal algorithm *selective search* takes approximately two seconds to generate 2000 proposal boxes on a modern CPU. Compared to the inference time of a convolutional neural network, this is an order of magnitude slower [55]. Additionally, it is not possible to jointly optimize the classification and proposal part of the model when using separate algorithms for the two parts.

To address this issue, Ren *et al.* [55] propose the *Region Proposal Network* (RPN), that again, shares computation with the underlying CNN to generate proposal boxes. An RPN operates in a fully-convolutional manner to regress bounding box offsets to predefined anchor boxes for every location in the shared convolutional feature map. Additionally, it outputs an *objectness score* for each of the regressed boxes.

The fully-convolutional network consists of an $n \times n$ convolutional layer, that extracts a d-dimensional feature vector from every location in the input feature map. This layer is followed by two sibling 1×1 convolutional layers. The first computes the $2k$ objectness scores (*i.e.*, the softmax probabilities of object *vs.* background), and the other computes $4k$ bounding box offsets (as defined in Equations (3.5) to (3.8)) with respect to k pre-defined constant anchor boxes for every position of the sliding-window of the $n \times n$ convolution. The anchor boxes are centered at the window and differ only in their scales and aspect ratios. Since the boxes are centered at every sliding window (resulting in a total number of $h_{\text{in}} w_{\text{in}} k$ anchor boxes for a single forward pass), the RPN is *translation invariant.* While Ren *et al.* [55] propose to use 3 different scales and aspect ratios — resulting in $k = 9$ anchor boxes — the choice of the right anchors is application-dependent. An illustration of a region proposal network can be found in Figure 3.2.

3.1.3 Combined Architecture

The combination of the RoI pooling layer and the RPN leads to the detection architecture proposed by Ren *et al.* [55]. The resulting network combines the three stages of the object detection pipeline (region proposal, region classification, and bounding box refinement) to a single network that is able to perform object detection by a single forward-pass through the network. Additionally, the network can be optimized in an end-to-end fashion without the need to temporarily cache features on the disk. An illustration of the architecture can be found in Figure 3.3. A big advantage of this architecture is the possibility to re-use

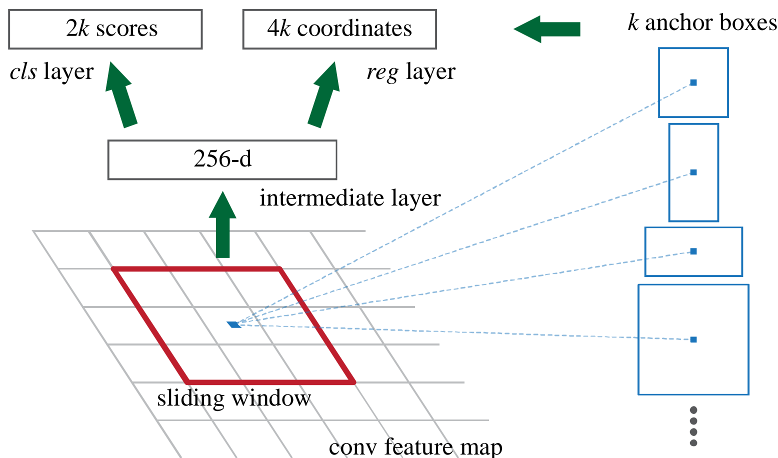


Figure 3.2: Illustration of an RPN with $n = 3$ and $d = 256$. At each spatial location of the convolutional feature map, the network generates an intermediate 256-dimensional feature vector. These features are further processed by the *cls* (class score) and *reg* (regression) layers, which are both 1×1 convolutional layers. Image taken from [55].

pre-trained classification networks (or at least parts of them) by replacing the last pooling layer with a RoI pooling layer.

The network has multiple outputs:

(1) A constant number of N top-scoring class agnostic region proposals encoded as an $N \times 4$ matrix, where each row corresponds to the values $(x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$ of the bounding box.

(2) The regressed, class-specific bounding box offsets in the form of an $N \times 4k$ matrix, where k is the number of classes and each row holds the bounding box offsets described in Equations (3.5) to (3.8) for each class.

(3) The predicted softmax class scores for every class, encoded by an $N \times k$ matrix.

The final detection results are given by the class scores and the corresponding region proposals after applying the regressed offsets and performing a greedy non-maximum suppression individually for each class.

3.2 Incorporating Depth Data

In the previous section, we summarized the general architecture of the object detection network. Since we are working with RGB-D data in this thesis, we will now describe how this architecture can be extended to fit this need.

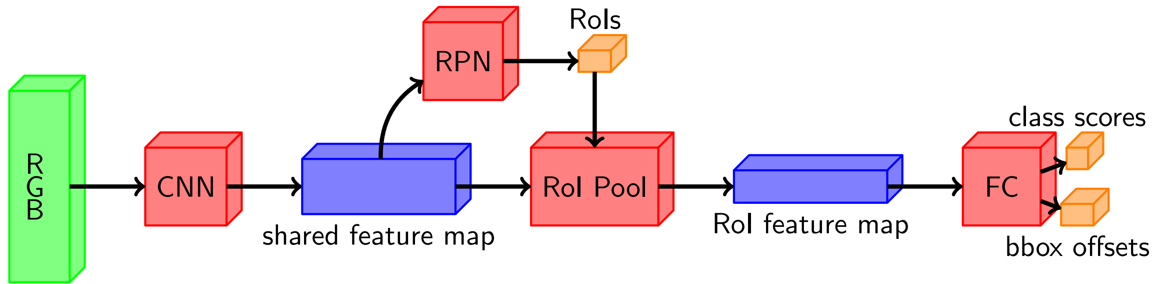


Figure 3.3: Illustration of the object detection architecture of *Fast R-CNN* [55]. The CNN block is a placeholder for an arbitrary sub-network producing a convolutional feature map. The RPN computes the region proposals, which are used by the RoI Pooling layer to pool fixed-size feature maps. The *Fully-connected* (FC) block, again, is a placeholder for a sub-network producing the class scores and class specific bounding box offsets for every region proposal.

One trivial solution is to adjust the input dimensions of the network in order to fit RGB-D images. However, this changes the dimensions of the architecture and requires to train the whole network from scratch. Thus, this approach demands a lot of training data and further leads to a significant computational effort.

Another approach is to train two separate networks — one for RGB, and one for depth images — and combine the results as in [25] and [26]. The main disadvantage of this approach, however, is that it requires to train the region proposal network separately, since the output of the network is not clearly interpretable if the networks were not operating on the same region proposals.

Our approach is to process both modalities in the same network, but without the need of changing the architecture of the underlying classification network. This requires some kind of modality-fusion, which we detail in the following.

3.2.1 Modality Fusion

In order to process RGB-D images with the architecture described in Section 3.1.3, we extend it by replicating the CNN and FC blocks. The original network processes the RGB modality, whereas the replicated network takes care of the depth modality. In order to get unified detection results from the two networks, we fuse the modalities inside the network. In this thesis, we investigate two different fusion approaches, which differ primarily in the location of the fusion.

Late Fusion The first approach fuses the modalities at the latest possible location in network, resulting in two almost independent modality streams. We call this approach *late fusion*. Given an RGB and the corresponding depth image, each modality is

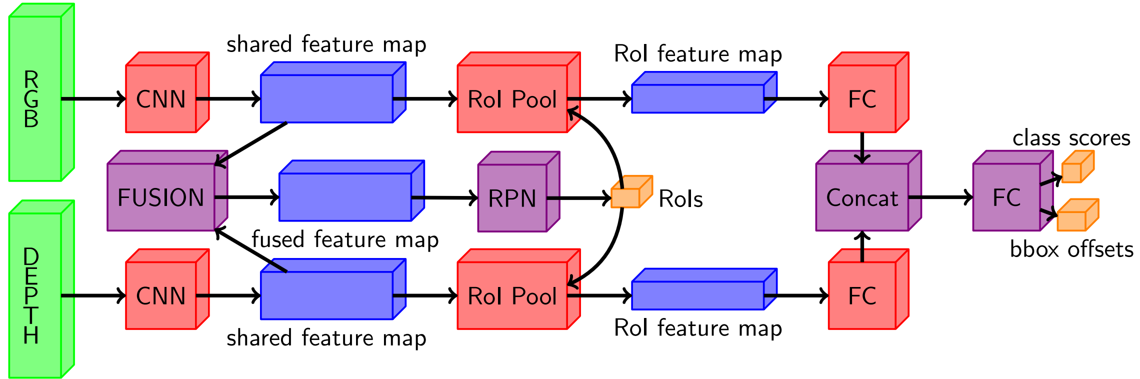


Figure 3.4: Illustration of the late fusion approach. RGB and depth data are processed independently until the last FC layer. An additional FC network produces the final output based on the concatenated features. The shared RPN has access to both modalities.

processed individually (including RoI pooling for both modalities). Instead of directly outputting the class scores and bounding box offsets, the output features of the two FC sub-networks are concatenated. An additional FC layer takes the concatenated features as input, and computes the final output of the network. To overcome the previously mentioned issue of different region proposals, the RPN is shared between the modalities. However, instead of using a pre-trained RPN operating only on one modality, we also fuse the convolutional feature maps produced by the two modality networks before feeding them to the RPN. The architecture of the late fusion approach is illustrated in Figure 3.4.

In fact, the late fusion approach is similar to the ensemble approach of Gupta *et al.* [25]. The difference lies in the weighting of the networks. Instead of engineering a weighting strategy, the weighting is implicitly learnt by the network. Additionally, the region proposal network has access to both, the RGB and depth data.

Early Fusion Late fusion consists of two full-sized classification networks, and since the modality streams are trained independently, the parameters cannot be shared between the layers. Especially the FC layers at the end of the network introduce a significant amount of parameters.

The second approach, which we call *early fusion*, minimizes the size of the network in terms of the number of trainable parameters. To that end, we cut down the network by replacing the independent parts after the RoI pooling layers by a shared RoI pooling layer operating on the fused feature map, *i.e.*, the one that is already forwarded to the RPN. The rest of the network is identical to the original architecture

described in Section 3.1.3, with the difference that the network operates on the fused features. An illustration of the early fusion network can be found in Figure 3.5.

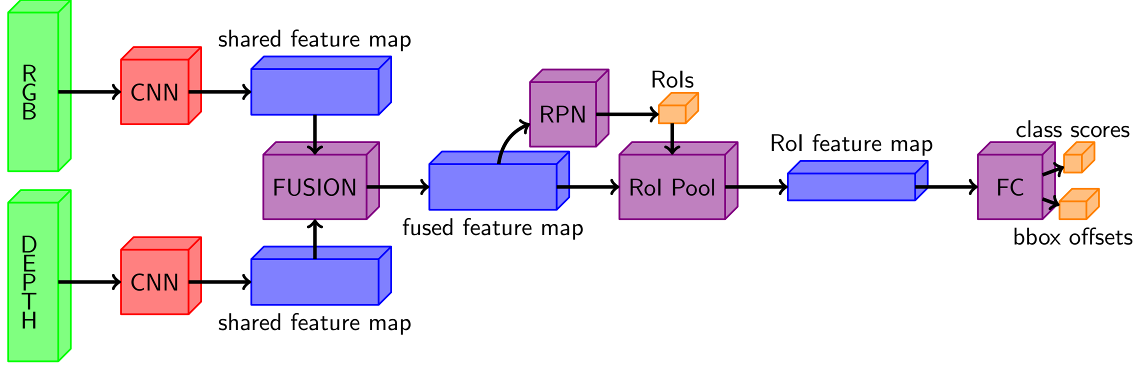


Figure 3.5: Illustration of the early fusion approach. In contrast to the late fusion approach, RGB and depth data are processed independently only in the convolutional part of the network. The convolutional feature maps are fused and act as input to both, the RPN and the RoI pooling layer. The number of parameters is significantly reduced compared to the late fusion approach.

Fusion Methods

The common part of both approaches is the fusion of the two convolutional feature maps. While in the late fusion approach, it only provides the input to the RPN, the early fusion approach relies on it as the sole source of fusion. In contrast to the late fusion, where the output features are already flattened by the FC layers, we must preserve the structure of the feature maps at this point.

Given the two feature maps $\mathbf{M}^{\text{rgb}}, \mathbf{M}^{\text{depth}} \in \mathbb{R}^{H \times W \times D}$, the output of the fusion must be another feature map $\mathbf{M}^{\text{fused}} \in \mathbb{R}^{H \times W \times D}$. We consider five different fusion techniques, namely (1) *average fusion*, (2) *sum fusion*, (3) *max fusion*, (4) *conv fusion*, and (5) *inception fusion*:

(1) **Average fusion** At every spatial location and for every feature in the input feature maps, compute the average as the fused feature $\mathbf{M}^{\text{fused}}_{h,w,d} = \frac{\mathbf{M}^{\text{rgb}}_{h,w,d}}{2} + \frac{\mathbf{M}^{\text{depth}}_{h,w,d}}{2}$.

(2) **Sum fusion** At every spatial location and for every feature in the input feature maps, compute the fused feature as the sum $\mathbf{M}^{\text{fused}}_{h,w,d} = \mathbf{M}^{\text{rgb}}_{h,w,d} + \mathbf{M}^{\text{depth}}_{h,w,d}$.

(3) **Max fusion** At every spatial location and for every feature in the input feature maps, take the max value of the two values as the fused feature $\mathbf{M}^{\text{fused}}_{h,w,d} = \max\left(\mathbf{M}^{\text{rgb}}_{h,w,d}, \mathbf{M}^{\text{depth}}_{h,w,d}\right)$.

(4) **Conv fusion** This method makes use of a 1×1 convolutional layer, in order to fuse the two feature maps. To that end, the channels of the feature maps have to be stacked at every spatial location as $\mathbf{M}^{\text{concat}}_{h,w} = \mathbf{M}^{\text{rgb}}_{h,w} \| \mathbf{M}^{\text{depth}}_{h,w}$. This intermediate feature map $\mathbf{M}^{\text{concat}} \in \mathbb{R}^{H \times W \times 2D}$ is processed by the convolutional layer, which outputs the fused feature map $\mathbf{M}^{\text{fused}} \in \mathbb{R}^{H \times W \times D}$.

(5) **Inception fusion** We also investigate a fusion method inspired by the inception module of GoogleNet [67]. As (4), this fusion operates on an intermediate feature map $\mathbf{M}^{\text{concat}}$ but instead of a single convolution, it consists of four parallel inception towers with 1×1, 3×3, and 5×5 convolutions and another tower with 3×3 max-pooling. The output features maps of each tower are again concatenated to form the fused feature map $\mathbf{M}^{\text{fused}} = \mathbf{M}^{\text{tower1}}_{h,w} \| \mathbf{M}^{\text{tower2}}_{h,w} \| \mathbf{M}^{\text{tower3}}_{h,w} \| \mathbf{M}^{\text{tower4}}_{h,w} \in \mathbb{R}^{H \times W \times D}$.

While average, sum, and max fusion are parameter-less, the parameters of the conv and inception fusion are learned during the training of the network.

3.2.2 Depth Encoding

Since we are using the same architecture for the RGB and the depth streams, the depth images must be encoded as already described in Section 2.6.2. We experiment with the HHA encoding proposed by Gupta *et al.* [25] and with a simple coloring of the *Height Above Ground* (HAG) values derived from the disparity maps. Please see Section 5.4 for details about the HAG-encoding.

3.3 Optimization

The final architecture consists of various parts, that can be optimized jointly using BackProp and SGD. The CNN, FC, and fusion blocks consist only of standard convolutional neural network layers. RoI pooling is implemented as a special layer and supports BackProp as well. Even the RPN consists only of convolutional layers. We will now describe the loss function of the network, which is required to train the network for the task of object detection.

3.3.1 Multi-task Loss

The network actually does multiple things at once: (1) compute objectness scores for the region proposals, (2) perform bounding box regression for the region proposals, (3) compute class scores for every region proposal, and (4) perform class specific bounding box refinement

for each region proposal. Naturally, the network makes errors for each task individually. Even though the only measure of interest during inference is the performance of the actual detection result (*i.e.*, the class scores and corresponding bounding boxes), more detailed information is necessary during training. Therefore, we compute individual losses $L_{\text{rpncls}}, L_{\text{rpnbbox}}, L_{\text{cls}}, L_{\text{bbox}}$ for each task. The actual loss of the network is the sum over all individual losses:

$$L = L_{\text{rpncls}} + L_{\text{rpnbbox}} + L_{\text{cls}} + L_{\text{bbox}}. \tag{3.9}$$

Score loss Both the objectness score and class score outputs of the network are softmax activations describing a discrete probability distribution over K classes (or object *vs.* background in the case of objectness scores). Given the corresponding ground truth scores from the dataset, we compute L_{rpncls} and L_{cls} as a cross-entropy loss of the predictions as defined in Equation (2.6).

Bounding box regression loss In contrast to score prediction — which is a classification task — the bounding box refinement is a regression task. Thus, a different loss function is needed. The standard loss function for regression tasks is the Euclidean loss, which computes the L_2 norm of the difference of the regressed outputs and the ground truth. In this thesis, we follow Girshick [22] and use a smoothed L_1 norm instead (in statistics, this loss is often called the Huber loss). The benefit of the L_1 norm is that it is less sensitive to outliers than the L_2 norm, and thus leads to fewer convergence problems. A visual comparison of the L_2 norm and the smoothed L_1 norm can be found in Figure 3.6.

Given the regression outputs $\mathbf{t}$ (which is a vector of bounding box offsets as defined by Equations (3.5) to (3.8)) and the corresponding ground truth vectors $\mathbf{v}$, we define the loss of a single example as

$$\sum_{i \in \{x,y,w,h\}} L_1 \left(\mathbf{t}_i - \mathbf{v}_i \right), \tag{3.10}$$

where

$$L_1(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| < 1, \\ |x| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{3.11}$$

Note that the loss of an example is ignored (*i.e.*, set to zero) if the class is *background* or *non-object*, respectively.

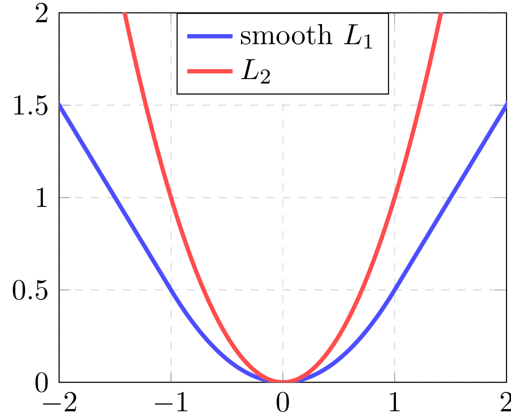


Figure 3.6: Plot of the smooth L_1 norm in comparison to the L_2 norm. Note that the smooth L_1 norm is less sensitive to outliers while maintaining the differentiability of the L_2 norm at 0.

3.3.2 Training Process

The original version of Faster R-CNN [55] proposes a stage-wise training process by alternating between the training of the RPN and the detection part of the network. This comes from the issue, that RoI pooling is not differentiable w.r.t. the region input. However, in additional material to their work, they note that it is also possible to train the whole network end-to-end, ignoring this issue. In this thesis, we stick to the end-to-end training process, which yields similar performance to the alternating optimization while converging nearly two times faster.

Initialization Most layers of the architecture can be initialized with the weights of a pre-trained classification network. We utilize a modified version of AlexNet [41], which was slightly improved and pre-trained on ImageNet by Zeiler and Fergus [77]. This model consists of five convolutional layers (including pooling and local response normalization layers), followed by two fully-connected layers.

Following [22], we use the convolutional layers — excluding the last pooling layer, which is replaced by the RoI pooling layer — for the CNN sub-networks (RGB and depth) in our architecture. The fully-connected layers are copied to the FC sub-networks. All remaining layers (*i.e.*, the fusion and rpn layers) are randomly initialized by the Xavier [38] initialization.

Mini-batch sampling The objective of the combined architecture is to share computation of the convolutional layers for all proposals. Therefore, the initial batch-size for the convolutional sub-network is 1.

The RPN generates $h_{\text{in}} w_{\text{in}} k$ anchors. We assign a binary class label (object or background) to each anchor, based on the *Intersection over Union* (IoU) of the anchors with the ground truth boxes. For each ground truth box, the anchor with the highest IoU overlap is marked as object. Additionally, all anchors which overlap more than 70% (*i.e.*, IoU > 0.7) with any ground truth box are positive samples as well. On the other hand, an anchor is marked as background, if the IoU is less than 0.3 for all ground truth boxes. For the loss computation, we randomly sample 128 positive and 128 negative anchors. The remaining anchors are ignored for the loss.

The detection part of the network takes all proposals generated by the RPN, and applies non-maximum suppression in order to reduce redundancy. 128 of the remaining proposals are sampled for the loss computation. A maximum of 32 batch samples are chosen as positive (*i.e.*, they correspond to an object of a specific class) if the IoU with a ground truth box is > 0.5. The rest of the batch is padded with negative background examples with $0.1 <$ IoU < 0.5.

4 Non-Maximum Suppression with Convolutional Neural Networks

Contents

4.1 Input Preparation **46**

4.2 Network Architecture **48**

4.3 Optimization **50**

In Chapter 3, we described our object detection pipeline with CNNs. However, since this pipeline — like most of the object detectors published recently — treats all detections independently from each other, it relies on a hard-coded post-processing procedure for NMS. This algorithm greedily merges detection boxes with IoU greater than a constant predefined threshold, keeping the boxes with the highest detection scores. Applying this greedy algorithm involves a trade-off between *recall* and *precision* by choosing a IoU threshold. Different threshold values must be evaluated on the validation set to find the best-performing one. By doing so, the results are heavily tuned to the validation set and thus, the detector may perform worse if the density and overlap of detections differ significantly at test time.

Hosang *et al.* [34] proposed a learnable alternative to the hard-coded NMS based on a CNN — called *Tyrolean network* (Tnet). Their network can be trained using the detections of an existing object detector only. No additional dataset labeling is needed. We re-use their ideas in this thesis, by putting a Tnet on top of our object detection network, which replaces the greedy NMS procedure and improves the performance of the detector in scenarios of different object densities.

4.1 Input Preparation

Tnet operates in a fully-convolutional manner. Therefore, the input as well as the output of the network is image-like data of arbitrary size. In order to provide meaningful data to the network, the detections are prepared and presented to the network in two different ways, which are (1) score maps encoding the position and the score of the detections, as well as the (2) IoU map encoding the overlaps of each detection and other detections in a local neighborhood.

4.1.1 Input Grid

Before computing these maps, the detections are mapped into a smaller two-dimensional grid. This grid is a down-sampled variant of the detection boxes' pixel coordinate system. The down-sampling factor is chosen in a way, such that the computational effort is reduced while the detection performance does not suffer. The coordinate system is divided into cells, where each cell corresponds to a $H_{\text{grid}} \times W_{\text{grid}}$ area of pixels in the detections' original domain. Therefore, for an input image of size $H \times W$, the detections will be mapped into a grid of size $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil$. The mapping $M : \mathbb{R}^4 \rightarrow \mathbb{R}^2$ of a bounding box $\mathbf{b}$ to grid coordinates $\mathbf{g}$ is given as

$$\mathbf{b} = \begin{pmatrix} b_{\text{xmin}} & b_{\text{ymin}} & b_{\text{xmax}} & b_{\text{ymax}} \end{pmatrix}^\top, \tag{4.1}$$

$$\mathbf{g} = \begin{pmatrix} g_{\text{x}} & g_{\text{y}} \end{pmatrix}^\top, \tag{4.2}$$

$$= M(\mathbf{b}) = \begin{pmatrix} \left\lfloor \frac{b_{\text{xmin}} + \frac{b_{\text{xmax}} - b_{\text{xmin}}}{2}}{H_{\text{grid}}} \right\rfloor \\ \left\lfloor \frac{b_{\text{ymin}} + \frac{b_{\text{ymax}} - b_{\text{ymin}}}{2}}{W_{\text{grid}}} \right\rfloor \end{pmatrix}. \tag{4.3}$$

In other words, the coordinates of a box center are mapped to the down-sampled grid. If more than one detection falls into the same cell of the grid, the highest scoring detection will be assigned to the cell and the others will be discarded.

Given this mapping, each spatial location in the input maps corresponds to at most one detection. Note that in contrast to the detector in [34] — which is a dense sliding window detector — our detector provides only a sparse set of detections. Hence many cells of the detection grid do not correspond to any detection. This sparsity poses additional challenges during training since the large amount of zero values reduces the variance in the

input maps, and hence might lead to significantly more overfitting. We describe how to compensate this issue in Section 4.3.1.

4.1.2 Input Maps

The input grid defines the spatial mapping of the detections and provides a correspondence between the original detections and the network's input. The actual input feature maps are based on the input grid, but represent the data in a way, that is processable by the network. Every location in each of the input maps contains information about a single detection, that helps the network to decide whether it should keep or suppress the detection. In the following, we describe the different input feature maps and their purposes.

Score Maps In order to encode the positions and the corresponding confidence scores of the detections, the network computes score maps. Each detection is represented in the raw score map by its score value at the mapped spatial location of the bounding box center in the input grid. Empty cells in the detection grid (*i.e.*, cells without a corresponding detection due to the sparse set of detections) correspond to a score of 0. Figure 4.1 illustrates how the original detections are encoded in the raw score map.

The task of the network is to mimic NMS with a dynamic IoU threshold for every detection. Since this requires the network to perform complex ranking operations, which are hard to learn for a CNN with convolutions and ReLU non-linearities only, auxiliary score maps are provided. These supplementary maps encode the confidence scores in the same way, but after applying traditional greedy NMS with different fixed thresholds. The set of thresholds is a hyper-parameter of the network and must be tuned during training.

The score maps are provided as a single input feature map, where each channel stores the score map for a specific NMS threshold. Therefore, the size of this feature map is $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil \times (T+1)$, where T is the number of NMS thresholds excluding 1 for the raw score map without any NMS applied to it.

IoU Map Additional context about each detection in form of IoU overlaps is provided by the IoU map. Therefore, we define a neighborhood of size $H_{\text{window}} \times W_{\text{window}}$ around each detection and compute the ratio between the area of intersection and the area of union for each bounding box in the neighborhood and the bounding box corresponding to the center cell of the neighborhood in the input grid.

The IoU feature map is of size $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil \times (H_{\text{window}} \cdot W_{\text{window}})$, where each channel corresponds to the IoU overlap of the reference bounding box in the center of the window and another bounding box from a cell in that window. The overlap with an empty cell is defined as 0.

Note that the ordering of the overlap values is not allowed to change between subsequent windows, since the network has to learn a correspondence between the score map and the IoU map. The (g_y, g_x, y, x) entry of the IoU map contains the IoU of the reference bounding box, centered on (x, y) with the bounding box that corresponds to the cell (g_x, g_y) where the IoU with an empty cell is zero. To process this tensor within CNN frameworks, we reshape it to 3 dimensions of size $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil \times (H_{\text{window}} \cdot W_{\text{window}})$.

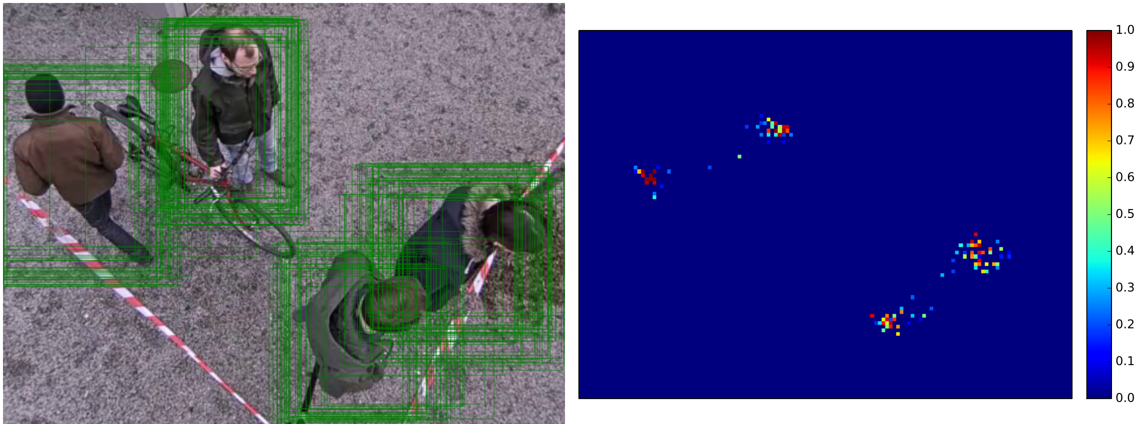


Figure 4.1: The boxes in the left image are the original detections of the detection model before NMS. The right plot shows the same detections mapped to the input grid in form of the raw score map. Note the sparsity of detections in the score map.

4.2 Network Architecture

As illustrated in Figure 4.2, the network consists of 4 stages with only convolutional layers. Thus, the network operates fully-convolutional and performs NMS in a single forward-pass without any post-processing. The network is divided into 4 sequential stages:

(1) The first stage consists of two sibling convolutional layers — one processing the score maps and the other one the IoU map. Both are followed by a ReLU non-linearity. These layers are important as they allow the network to put the two different sources of information into context. The score maps are processed by a $H_{\text{window}} \times W_{\text{window}}$

convolution, coinciding with the neighborhood window previously used to compute the IoU map. The latter, on the other hand, is processed by a 1×1 convolution, so both layers operate on information about the same set of neighboring detections in the input grid. Consequently, the produced feature maps contain highly correlated features at every spatial location.

(2) This correlation is further exploited in the second stage of the network, where the channels of the feature maps are concatenated and processed by another 1×1 convolution with ReLU activation. At this point, the different sources of information are fused for further processing.

(3) The third stage consists of another 1×1 convolution followed by a ReLU non-linearity. This stage may be extended to multiple convolutional layers, if the dataset requires a more complex model.

(4) The final stage of the network is again a 1×1 convolution. Since this layer is intended to output the final re-scored detections in form of a $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil \times 1$ score map, a sigmoid activation is used instead of a ReLU. The remaining scores of the output feature map correspond to re-scored confidence values of the final detections, so every remaining score > 0 is meant as an unsuppressed detection. The original bounding box can be found by the initial mapping of the detections to the input grid.

Except for the layer that operates on the IoU map, all convolutions are of size 1×1. This enables a computationally efficient evaluation of the network and ensures that the decision of whether a detection should be suppressed or not depends only on the detections in the configurable neighborhood window.

An important parameter of this architecture is the number of filters used by the convolutional layers. While the authors of the original paper [34] proposed to use 512 filters for each convolutional layer, we found that this number can be reduced for our purpose. Details about our findings can be found in Section 5.5.

Since the whole architecture can be implemented by means of standard CNN layers, we can combine Tnet with our detection network from Chapter 3, which enables inference of the combined network with a single feed-forward pass. To this end, we need to introduce a data layer which prepares the input feature maps for Tnet directly from the output of the detector as described in Section 4.1.2.

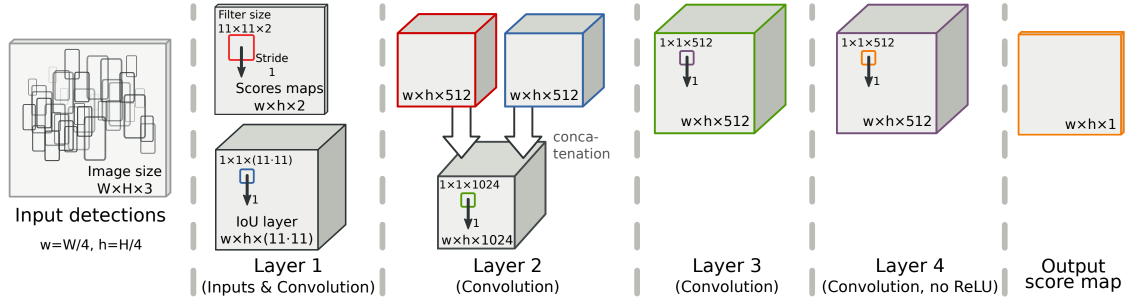


Figure 4.2: Architecture of a Tnet. Inputs are the detection boxes of the underlying detector. Each box corresponds to a convolutional feature map with its dimensions declared at the bottom. The squares inside the boxes corresponds to the convolution filters. Image courtesy of [34].

4.3 Optimization

Since the entire architecture consists only of standard CNN layers, it is possible to optimize the network layers with BackProp and gradient based optimization algorithms. We train Tnet by pre-computing the detections from our modified Faster R-CNN detector. A custom data layer reads the detections from the file system and prepares the input feature maps.

4.3.1 Training Targets and Loss

Since the objective of Tnet is to perform NMS, we have to match detections to ground truth boxes such that each ground truth box is assigned to at most one detection box. The matched detection boxes serve as positive samples, whereas all other detections are negative examples. In this way, the network is tuned towards choosing a single detection box for a unique object based on the observations it gets from the neighboring scores and overlaps.

The matching is done for every ground truth box by first rejecting all detections that overlap less than 50 % (*i.e.*, with IoU < 0.5). From the remaining detections, the highest scoring one is chosen as the positive example for the current ground truth. In doing so, we get a score label for every cell in the input grid. Having these labels, we can construct the target score map with dimensions $\left\lceil \frac{H}{H_{\text{grid}}} \right\rceil \times \left\lceil \frac{W}{W_{\text{grid}}} \right\rceil \times 1$.

Since the network's outputs can be interpreted as probabilities — just like the score output of the detection network but structured in a 2-dimensional grid — we follow [34] and use a cross-entropy loss (as defined in Equation (2.6)) to perform gradient-based optimization. Additionally, the different loss terms are weighted such that the weight of all true positive detections matches the weight of all background detections in each mini-batch.

To compensate the lack of dense detections in the input grid, we further set the weights at cells without any detection in the $H_{\text{window}} \times W_{\text{window}}$ neighborhood to zero. Without this filtering, the network does not converge due to a surplus of empty detection windows. The weights are set to $\frac{N_{\text{total}}}{2N_{\text{bg}}}$ for background detections and to $\frac{N_{\text{total}}}{2N_{\text{fg}}}$ for true positives, where N_{total} is the total number of cells that are affected by the filtering described before, N_{bg} the number of background detections, and N_{fg} the number of foreground detections, respectively.

Unfortunately, it is likely that the training input contains certain faulty detections — simply because it is generated by a learned detector itself. It is challenging for the network to differentiate between false positives on the background and true positives. Also instances with high overlap can be a problem when the confidence of the detection is low because of severe occlusion. The weight of such hard examples are lowered by a factor of 0.3. Hard examples are background detections that are not suppressed by a NMS threshold of 0.3 and true positives that are suppressed by the same threshold.

4.3.2 Initialization and Solver

The network is trained from scratch without any pre-training. Therefore, we initialize the weights of the convolutional layers with MSRA [31] and train the model with the *Adam* solver [40] as suggested in [34]. Additionally, we perform gradient clipping [51].

4.3.3 Data Augmentation

In order to improve generalization and to compensate for small datasets, we employ several data augmentation techniques during training. In contrast to data augmentation in the image domain, which is mostly done by image transformations, for Tnet the inputs are detection boxes and their corresponding detections scores. We experiment with three different augmentation approaches and their combinations:

(1) **Flipping** The simplest rescoring approach is to flip all detections both vertically and horizontally. In this way, the relative positions of the detections stay the same and we are able to increase the amount of training samples by a factor of 4.

(2) **Rescoring** To simulate noise in the confidence scores of the detector, we draw a value from a normal distribution with zero mean and a standard deviation of 0.1 for each detection and add the value to the original score. The augmented score will be clipped to the range $[0, 1]$.

(3) **Moving** We randomly sample translation vectors for each detection box from a normal distribution with zero mean and a standard deviation of 6 pixels and apply the translation to the box. By doing so, each box is expected to move one cell in the input grid both horizontally and vertically.

5 Experiments and Evaluation

Contents

5.1 Performance Metrics 54
5.2 Datasets 56
5.3 Implementation Details 58
5.4 Depth Fusion for Pedestrian Detection 59
5.5 Learning Non-Maximum Suppression 66
5.6 Runtime Performance 70

In this chapter, we present the performance of our pedestrian detection pipeline. Due to the special setting of the scenes in which our detector operates (see Section 1.1 for details), a comparison to other state-of-the-art pedestrian detectors is not possible and would not yield much information. Instead we evaluate the detector on our custom datasets that were recorded and labeled for this thesis. In order to get a better understanding of the performance, our experiments focus on the contribution of the various parts of our models and of the techniques used during optimization.

We start this chapter with a description of the performance metrics used throughout our experiments followed by a description of the datasets and some implementation details. Finally, we discuss our experiments and the corresponding findings with the detection and NMS models from Chapter 3 and Chapter 4. Both qualitative and quantitative evaluations will be presented.

		ground truth	
		object	no object
detector	object	TP	FP
	no object	FN	TN

Table 5.1: Illustration of TP, TN, FP, and FN based on ground truth annotations and detector decisions in form of a confusion matrix.

5.1 Performance Metrics

For quantitative analysis of a detector's performance, we need reliable performance metrics. In case of binary classifiers — like our detector — the output can be split into four categories:

***True Positive* (TP):** The sample is *correctly* classified as *object*.

***True Negative* (TN):** The sample is *correctly* classified as *background*.

***False Positive* (FP):** The sample is classified as *object by mistake*.

***False Negative* (FN):** The sample is classified as *background by mistake*.

An illustration of the relations between the detector's output and the ground truth can be found in Table 5.1 by means of a confusion matrix.

Naturally, we want the number of TP and TN to be high, whereas the number of FP and FN should be low. For a perfect classifier which makes no mistakes, the sum of FP and FN would be zero. On the other hand, the number of TP and TN would depend on the dataset.

In practice, the uncorrelated numbers of TP, TN, FP, and FN are hard to interpret. Therefore, different metrics based on these raw values are used to compare the performance of classifiers. In the following, we will introduce the metrics used throughout this chapter.

5.1.1 Precision and Recall

Two commonly used metrics for object detectors are *precision* and *recall*. Precision measures the relevancy of the detections by correlating the number of TP with the number of all

detections. Formally, it is defined as

$$P = \frac{T_p}{T_p + F_p}, \tag{5.1}$$

where T_p is the number of TP and F_p is the number of FP. Recall, on the other hand, is a measure of how much of the relevant objects are actually found by the detector and is given by

$$R = \frac{T_p}{T_p + F_n}, \tag{5.2}$$

where F_n is the number of FN.

Good performing detectors yield high values for both, R and P, indicating that it detects most of the objects in the dataset correctly and, at the same time, makes few mistakes. The relationship between the two values can be visualized by a precision *vs.* recall plot where precision is plotted as a function of recall.

5.1.2 Average Precision

While precision and recall can be visualized and compared by means of the precision *vs.* recall plot, we still do not have a single value describing the performance of the detector. To that end, we also compute the *Average Precision* (AP) of the detector which is the *Area under the Curve* (AuC) of the precision *vs.* recall plot.

We follow [18] and compute the mean of the interpolated precision at eleven equally spaced recall levels as

$$\mathrm{AP} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots, 1\}} P_{\text{interp}}(R), \tag{5.3}$$

$$P_{\text{interp}}(R) = \max_{\widetilde{R} \geq R} P(\widetilde{R}). \tag{5.4}$$

where $P(\widetilde{R})$ is the measured precision at recall $\widetilde{R}$.

5.1.3 Proposal Coverage

In order to evaluate the quality of the region proposals generated by the RPNs of our models, we compute the *coverage* of the ground truth regions as a function of the number of region proposals. This metric does not only count how many of the ground truth boxes are covered, but also to what extent the region proposals match the ground truth boxes.

This is achieved by computing the maximum IoU between the ground truth boxes and the proposal boxes.

We follow [25] and define the coverage as

$$\text{Coverage}(K) = \frac{1}{C}\sum_{c=0}^{C}\left[\frac{1}{N_c}\sum_{j=1}^{N_c}\max_{k\in[1\dots K]}\text{IoU}(P_k^{i(c,j)}, G_j^c)\right], \tag{5.5}$$

where C is the number of classes, N_c the number of object instances for class c, $\text{IoU}(x,y)$ is the IoU of region x and y, $P_k^{i(c,j)}$ is the kth proposal ranked by its score in the image with index $i(c,j)$ containing the jth instance of class c, and G_j^c is the the ground truth region of the jth instance of class c.

5.2 Datasets

In this section we describe the two datasets used in our experiments. The *Campus* and *Vienna* datasets consist of frames sampled from videos recorded using a stereo setup. The cameras were mounted on a telescopic rod and a traffic light pole, respectively, pointing downwards onto the scene. Disparity images are pre-computed using a GPU implementation of [62] after synchronizing and rectifying the frames of the two cameras. The final datasets comprise pairs of images from the left camera and the corresponding disparity images — each of size 640×480. The datasets provide bounding box annotations for each pedestrian in the scene.

Campus This dataset was recorded in a controlled environment. It is based on approximately 20 minutes of video material showing nine different people walking through the scene. It includes very crowded situations with many people walking or standing near each other as well as people walking alone. Additionally, some border-cases such as people wearing umbrellas and hats are included. As the number of people is very limited in this dataset, it is not well suited for training. Instead we use it to test the generalization capabilities of our models.

The dataset includes 872 annotated frames containing 2301 different annotations. The average number of annotations per frame is ≈ 2.6. Example images of this dataset can be found in Figure 5.1.

Vienna In contrast to the previously described *Campus* dataset, this dataset was recorded in a realistic environment on a public site in Vienna. The recordings comprise

approximately 86 minutes of video material. Besides many different people, this dataset includes several negative samples along with dogs, barrows and trolleys. For that reason, this dataset is more eligible for training purposes.

We sampled 752 frames in which we annotated a total number of 2254 people with an average of ≈ 2.9 annotations per frame. Additionally, we paid attention to include significantly overlapping (*i.e.*, overlaps of more than 30%) bounding boxes. Figure 5.2 shows examples of images of the *Vienna* dataset.

A peculiarity of these datasets is the large amount of cut-off pedestrians due to the top-down viewpoint of the cameras, *i.e.*, while entering and existing the field-of-view only the legs are visible. To compensate this issue, the annotated bounding boxes were chosen such that parts of them are allowed to lie outside of the visible image. In this way, the bounding box regressor gets more uniform targets during training. For a fair evaluation, we clip the predicted and the ground truth bounding boxes to the image boundaries at inference time.

Figure 5.1: Two examples from the *Campus* dataset.

KITTI In addition to the *Campus* and *Vienna* datasets, we also use parts of the KITTI object detection dataset [21]. The object detection dataset consists of 7481 annotated training images and 7518 test images of traffic scenes. The images were recorded using a complex setup of multiple cameras and sensors mounted on the roof of a car. Since the internal and external camera parameters are provided as well, we are able to compute disparity images in the same way as for our own datasets. Due to the different resolution of 1392 × 512 compared to the other datasets, we vertically cut each image (and the corresponding annotations) into halves and keep all halves

Figure 5.2: Two examples from the *Vienna* dataset.

containing at least one pedestrian annotation. This results in 1813 images of 696×512 pixels, which we can use in our experiments.

Pedestrians in this dataset occur only sparsely. Further, since the camera is mounted on top of a car, pedestrians are recorded from the front, back, or side view, but not from above. Hence, we do not evaluate our models on the KITTI dataset, and use it to bootstrap the models during training in order to improve generalization of our models. This is necessary, since the *Campus* and *Vienna* datasets both provide only still images with little to no variance in the background.

5.3 Implementation Details

In this section, we provide some details about the implementations of the models, the experiments, and the evaluation environment. We also describe aspects of the training and testing procedures that are common to all experiments.

5.3.1 Frameworks and Code

We use the Caffe deep learning framework [38] for all our CNN implementations. The framework is configured to perform most of its computation on the GPU. We also enabled cuDNN [8] support for additional performance acceleration on Nvidia® GPUs.

The code of the object detection network is based on the publicly available Python™ implementation of Faster R-CNN [55], which was adjusted to fit our needs. Tnet was implemented from scratch as no public code was available. The evaluation code is based on the Matlab® database tools of Pascal VOC [18] but was ported to Python™.

5.3.2 Hardware Setup

All experiments and evaluations were performed on the same machine. The main components are an Intel® Core™ i5-4570 CPU with 4 physical cores, 16 GB memory, and an Nvidia® GeForce® GTX 970 GPU with 4 GB memory.

5.3.3 Training and Testing Procedure

Some aspects of the training and testing procedures are common to all experiments. This includes input preparation, data augmentation and common parameter choices:

Input preparation We compute the mean pixel values for each channel of the input images in our datasets and subtract them from the inputs of the CNNs in order to get approximately zero mean for each channel. The channels of the input images are in BGR order, since the pre-trained models, which we used for fine-tuning, were trained this way. Additionally, we scale each input image such that its longer side is 1000 pixels wide.

Data augmentation In order to increase the number of training samples, we extend the datasets by adding a vertically flipped version of each image. Note that other data augmentation such as random cropping would be possible, however, this is implicitly done by the RoI pooling layer during training.

SGD parameters Unless stated otherwise, we set momentum to a value of 0.9 and the weight decay regularization factor to $5 \cdot 10^{-4}$ for all experiments. These values result in stable convergence for many applications [17, 22, 55, 63], including our experiments.

Inference parameters Additional hyperparameters used during inference like the NMS-threshold are chosen per model such that the AP on the validation set is maximized. To allow for a fair qualitative comparison, we choose the detection threshold (*i.e.*, the minimum score of a positive detection) such that all models operate on either the same precision or recall level.

5.4 Depth Fusion for Pedestrian Detection

The experiments in this section deal with the question on how the additional depth information improves the detection performance over RGB-only images. Therefore, we compare the different fusion architectures from Section 3.2 and show how tweaks in the

training procedure can improve the performance and generalization. Additionally, we evaluate the impact of the depth features on the region proposal generation with an RPN.

As already noted before in Section 3.2.2, we explore different depth encodings, *i.e.*, (1) HAG-coloring and (2) Gupta *et al.*'s [25] HHA-encoding which was already described in Section 2.6.2. In contrast to (2), where each channel of the image corresponds to a different property computed from the depth values, (1) simply colors the HAG values using MATLAB®'s default colormap. We compute the HAG values by fitting the ground plane of each scene in the datasets using *Random Sample Consensus* (RANSAC) [20] and computing the point-to-plane distance for each point given by the dense depth maps. Since we want to detect pedestrians, the maximal HAG value is chosen to be 3 m.

The training process follows [55] as described in Section 3.3.2 with some minor adjustments. First, since the ground truth boxes in our datasets can lie partly outside the image boundaries, we adapt the anchor/proposal to ground truth matching by clipping the ground truth to the image boundaries for the overlap computation. This is necessary since RoI pooling can not be done outside the image. Second, we do not compute the overlap as IoU. Instead we use a slightly different measure

$$\frac{|X \cap Y|}{\min(|X|, |Y|)}, \tag{5.6}$$

where X and Y are bounding boxes, $|X \cap Y|$ is the area of intersection, and $\min(|X|, |Y|)$ is the smaller area of X and Y. During training, this measure is beneficial compared to the IoU criterion. It helps reducing false positive pedestrian part predictions, such as legs or heads. With this min-area criterion, these predictions are more likely to be suppressed by stronger full-body detections compared to the IoU criterion.

All models in this section are trained on the training set from the *Vienna* dataset. Note that no image from the *Campus* dataset is used during training. Instead, this dataset is used to test the generalization capabilities of the models.

Due to the small number of training samples, the static background in the training images, and the resulting overfitting concerns, we train the models without the last fully-connected layer just before the output layers (*i.e.*, we use *fc6* of the Zeiler Fergus network [77]). As our experiments show, this significantly reduces the amount of parameters without reducing the accuracy of the detector.

Model	**AP**	
	Early	**Late**
RGB	87.20 % (0.10)	
Depth	80.30 % (1.50)	
Sum fusion	89.00 % (0.80)	89.20 % (0.00)
Average fusion	88.80 % (0.70)	**89.75 %** (0.05)
Max fusion	89.05 % (0.35)	88.10 % (0.00)
Conv fusion	**89.75 %** (0.25)	87.55 % (0.85)
Inception fusion	87.65 % (0.85)	—

Table 5.2: Quantitative comparison of the baseline RGB and depth models with different early and late fusion models on the *Vienna* test-set. The depth images were encoded using HAG-coloring. The presented AP values are the average values of two independent trainings with randomized initialization. The values in parenthesis denote the standard deviation.

5.4.1 Quantitative Analysis

In order to compare the performance of the different models, we present the AP values on the *Vienna* test set in Table 5.2. The AP values are averaged over two independent trainings with random initialization and HAG-colored depth images.

The results show that all fusion models outperform the baseline RGB model by approximately 1.5–2.5 %. In the early fusion case, the convolutional fusion works best whereas in the late fusion case, it only achieves a slight performance boost of 0.45 %. We hypothesize that the late fusion model in combination with convolutional fusion is more prone to overfitting, since the number of parameters is larger compared to sum, average, and max fusion. The lower performance of the early fusion model with inception fusion further supports this claim, since it also introduces more parameters. We hypothesize that more powerful fusion approaches such as inception fusion or convolutional fusion could lead to performance improvements on larger datasets.

Although the *Vienna* test set does not contain any person present in the training set, the images capture the same background because they are sampled from video sequences of the same scene. In order to get a better understanding of the generalization capabilities of the model, we show the performance on the *Campus* test set in Table 5.3. Additionally, we provide precision *vs.* recall curves in Figures 5.3 and 5.4.

Again, we can see that the fusion models strictly outperform the baseline RGB model, but — compared to the *Vienna* test set — by a larger margin of $\approx$ 3.7–8 %. The low performance of the convolutional fusion (with both, early and late fusion) and the slightly

Model	**AP**	
	Early	**Late**
RGB	81.95 % (0.35)	
Depth	52.05 % (3.55)	
Sum fusion	88.60 % (1.00)	87.55 % (0.65)
Average fusion	87.00 % (0.00)	**87.70 %** (0.90)
Max fusion	**89.89 %** (0.20)	87.65 % (0.75)
Conv fusion	86.35 % (0.55)	85.60 % (1.10)
Inception fusion	88.85 % (0.85)	—

Table 5.3: Quantitative comparison of the baseline RGB and depth models with different early and late fusion models on the *Campus* test set. The depth images were encoded using HAG-coloring. The presented values AP values are the average values of two independent trainings with randomized initialization. The values in parenthesis denote the standard deviation.

worse performance of the late fusion models emphasize our previous observation that the models overfit to the training data with increasing numbers of parameters.

If we further compare the performance of the early fusion models on both datasets, it seems that max and sum fusion perform better than average fusion in general. However, again it is not clear if this would hold for larger datasets as well.

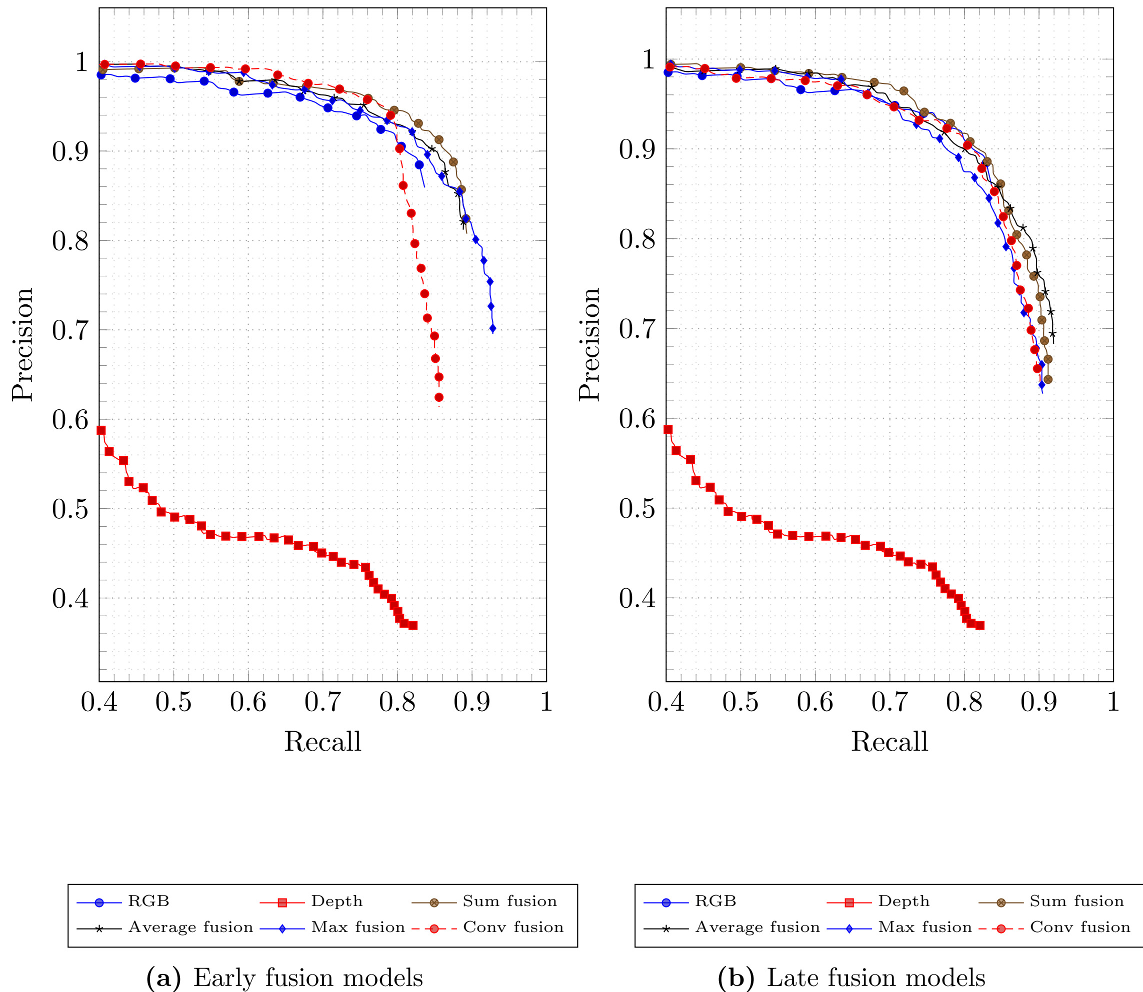
Looking at the precision *vs.* recall plots, we see that all early fusion models (except for the convolutional fusion) yield a higher precision than the RGB baseline at all recall levels. The late fusion models, on the other hand, do not really improve precision. However, they still improve over the baseline by achieving higher recall values.

HHA-Encoding

In order to compare HAG-coloring and HHA-encoding, we re-trained the depth model as well as the best performing fusion models with HHA-encoded depth images. Table 5.4 shows the evaluation results on the *Campus* test set. Although the fusion models are able to outperform the RGB model, the overall performance is worse than the HAG models. We suppose that the additional channels (*i.e.*, horizontal disparity and angle of gravity) — while they may be beneficial in indoor scenes with many discontinuities and small structures — are not very expressive or even harm the performance in the scenes we are targeting.

KITTI Bootstrapping

To prevent overfitting to the restricted scenes in our datasets, we bootstrap the models during training with images from the KITTI dataset. To that end, we randomly sample



(a) Early fusion models **(b)** Late fusion models

Figure 5.3: Precision *vs.* recall curves of the depth fusion models evaluated on the *Campus* test set.

Model	**AP**	
	Early	**Late**
RGB	81.95 %	
Depth	77.10 %	
Average fusion	–	**84.90 %**
Max fusion	**84.80 %**	–

Table 5.4: Quantitative comparison of the baseline RGB and depth models to the best fusion models with HHA-encoding.

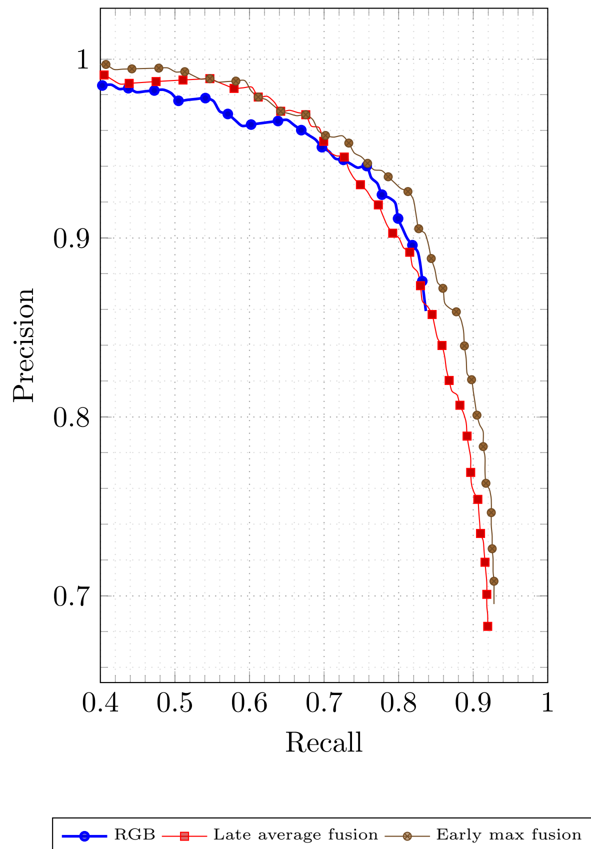


Figure 5.4: Precision *vs.* recall comparison of the best-performing early and late fusion models on the *Campus* test set.

images from the actual training dataset and KITTI such that both datasets are equally present during training. The intuition is to prevent the network from overfitting to the static background in the *Vienna* dataset. We hypothesize that KITTI is a good choice because of the similar setting and the opportunity to compute depth maps with the same stereo algorithm.

Our experiments show that the evaluation results are not heavily influenced by KITTI bootstrapping. We hypothesize that, although the additional background variation could be a benefit, the overall appearance of pedestrians (*i.e.*, frontal and side-view) in KITTI is significantly different compared to our dataset (top-view). In combination with the limited amount of training samples, the network is unable to learn a common pedestrian representation. Further, due to memory limitations, each mini-batch consists of only a single image. Since both tasks differ significantly, this might hurt neural network performance. Using multiple images per batch could improve this.

5.4.2 Qualitative Analysis

We further show qualitative comparisons between the best-performing RGB detector and the best-performing RGB-D detector (early max fusion method). To allow for a fair comparison, we choose the detection threshold for each model independently such that both operate on the same precision level of ≈ 0.9. Figure 5.5 shows some example images from the *Campus* test set.

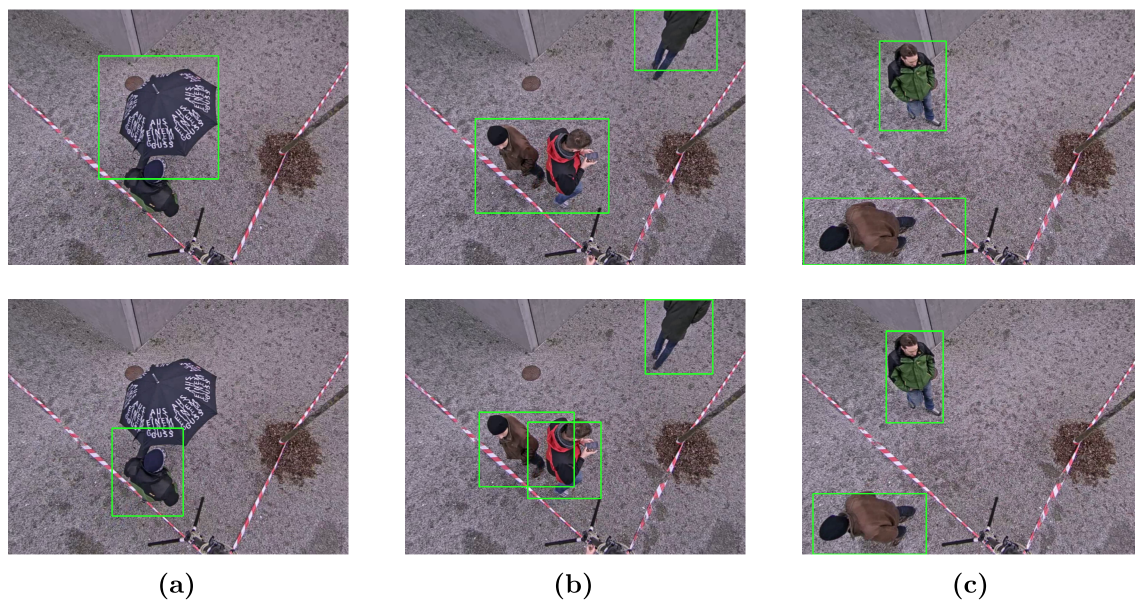


Figure 5.5: Qualitative comparison of the RGB baseline model (first row) and the best performing fusion model (second row). Both models operate on the same precision (0.9).

The first sample (a) shows that the RGB network gets confused by the umbrella, while the fusion network is able to correctly detect the person beside it while ignoring the umbrella. Note that the training set does not contain a person with an umbrella, emphasizing the observation of better generalization capabilities from quantitative analysis. Looking at the visual appearance in RGB space, one could argument that the umbrella in combination with the legs and the nearby person looks similar to other examples in the training set, which appear elongated due to the viewpoint (*e.g.*, Figure 1.3, bottom row, last two samples from the right). However, in depth space, these examples have a clearly distinguishable signature, which improves the predictions of the fusion model in such cases.

In the second sample image (b), the RGB network fails to correctly detect the two people standing near each other. Instead a single detection spanning both is returned. The fusion network, however, yields two detections.

The last sample (c) highlights improvements regarding the bounding box regression step. While the RGB model outputs a fairly large bounding box for the person in the bottom-left corner of the image, the fusion network yields a more accurate bounding box prediction. The same phenomenon — even if not that evident — can be observed at the bounding boxes of the other person in the image and the second sample.

5.4.3 RPN with Depth

In the previous sections we have shown how the fusion networks are able to outperform RGB-only networks. However, it is not perfectly clear at which point the additional depth data contributes to the performance boost.

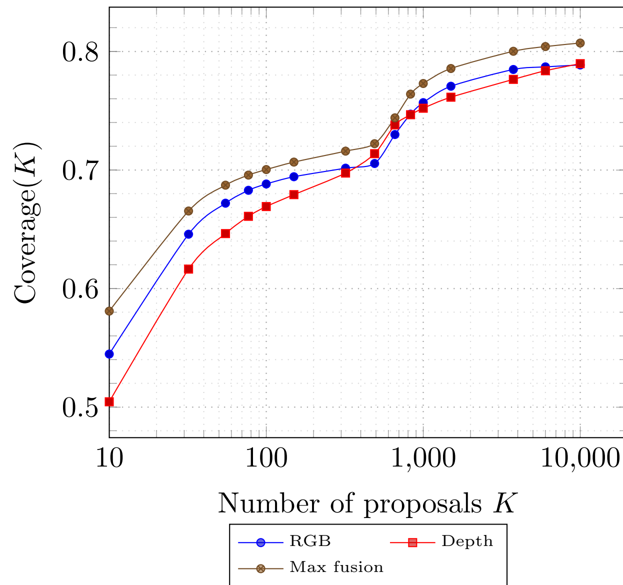


Figure 5.6: Comparison of the region proposal coverage between RGB-only, depth-only, and a max fusion RPN.

One crucial part of the model is the RPN which generates region proposals for further classification. Figure 5.6 compares the coverage of the region proposals generated by the RGB-only, depth-only, and max-fusion networks. We can see that the max-fusion network outperforms the single modality networks for every number of region proposals.

5.5 Learning Non-Maximum Suppression

In this section, we describe our experiments with the NMS network Tnet (recall Chapter 4). We compare its performance on our test datasets with the traditional greedy NMS algorithm

Model	AP		
	All	Overlapping	Non-overlapping
Tnet	**90.10 %**	**87.00 %**	**95.90 %**
NMS 0.9	41.20 %	37.30 %	49.40 %
NMS 0.8	67.80 %	61.80 %	76.40 %
NMS 0.7	85.60 %	78.10 %	93.40 %
NMS 0.6	**89.70 %**	**82.30 %**	95.40 %
NMS 0.5	88.30 %	81.00 %	**95.90 %**
NMS 0.4	87.10 %	79.30 %	95.30 %
NMS 0.3	86.30 %	77.90 %	95.20 %
NMS 0.2	83.30 %	74.00 %	95.00 %
NMS 0.1	78.20 %	65.40 %	94.30 %

Table 5.5: Quantitative comparison between traditional greedy NMS and our trained Tnet model. AP values are presented for the entire dataset (first column), images with at least one overlapping ground truth box (Overlapping), and images with no overlapping ground truth boxes (Non-overlapping).

used by the models in Section 5.4. The experiments show that Tnet is able to eliminate the need of choosing a specific threshold while improving performance.

The model is trained as described in Section 4.3 for $5 \cdot 10^4$ iterations on the *Vienna* training dataset. We augment the data by flipping, moving, and rescoring the raw detections of the best-performing model in Section 5.4 (*i.e.*, the early max fusion model). For the score map computation, we use a wide range of greedy NMS thresholds (*i.e.*, 1.0, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1). The size of the detection neighborhood is 11×11 and each cell in the input grid corresponds to 6×6 pixels of the original pixel-space. In this way, the rescoring context for each detection consists of all other detections lying in a window of 66×66 pixels around it in the original image.

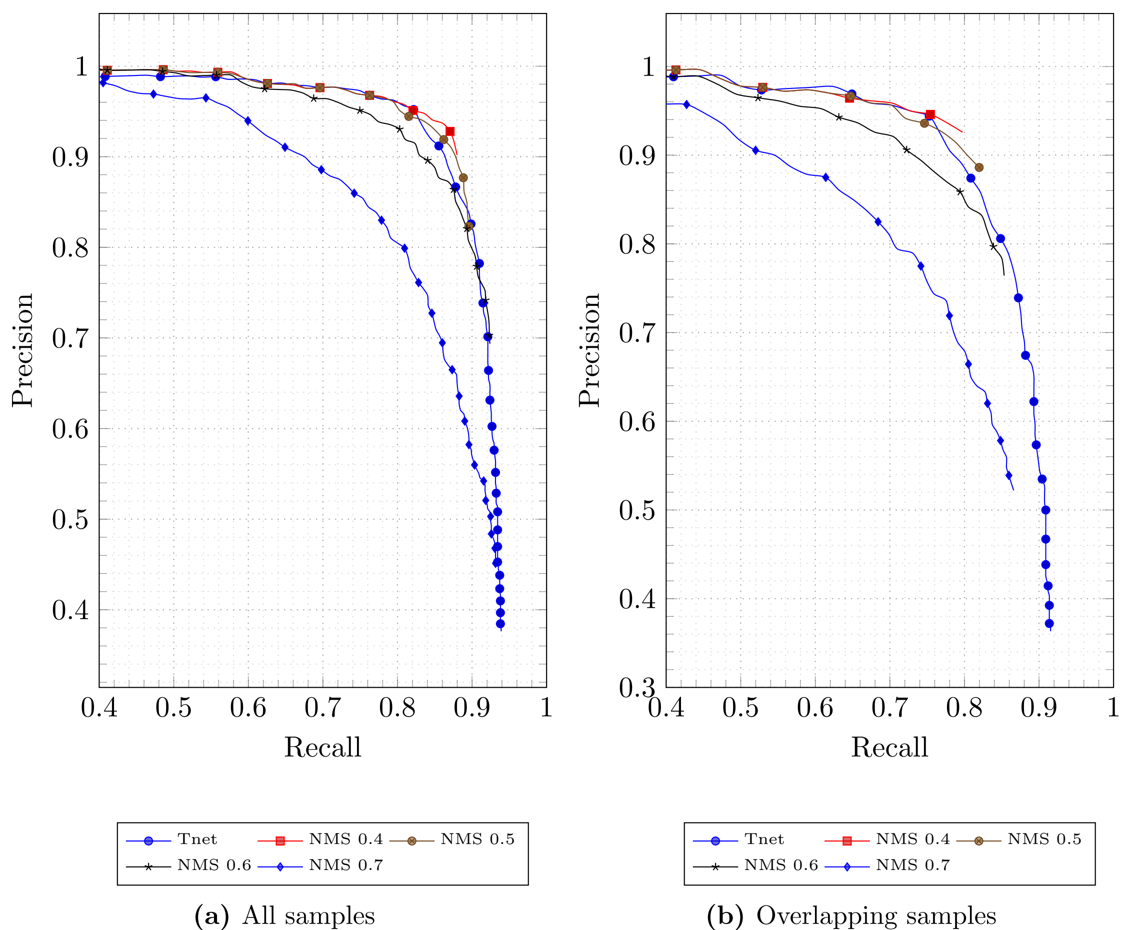
We use the *Adam* solver with a learning rate of 10^{-4}, a weight decay of $5 \cdot 10^{-4}$, and 0.9 momentum. Additionally, we perform gradient clipping such that the L_2 norm of all gradients does not exceed a value of 1000.

5.5.1 Quantitative Analysis

For a quantitative comparison between the best performing greedy NMS thresholds and Tnet we show the performance of the models in terms of AP in Table 5.5. Note that we want to evaluate the post-processing of the detector output and not the detection performance. Thus, we can combine the test sets of the *Vienna* and *Campus* datasets for this experiment which allows for a larger test set. We split the datasets into samples with (1) at least two

overlapping ground truth boxes and (2) without any overlapping ground truth boxes and evaluate the performance for both splits independently.

The first column shows that Tnet just slightly outperforms the best NMS threshold (= 0.6) on the entire dataset. If we only look at the samples with overlapping ground truth boxes (second column), we see that Tnet improves AP by 4.7 % compared to greedy NMS. For non-overlapping samples (third column), Tnet is on par with with the best-performing NMS threshold. However, while a threshold of 0.6 performs best for the overlapping samples, for non-overlapping samples a threshold of 0.5 is slightly better. This illustrates how Tnet can overcome the need to choose a specific threshold which is one of the disadvantages of greedy NMS.



(a) All samples **(b)** Overlapping samples

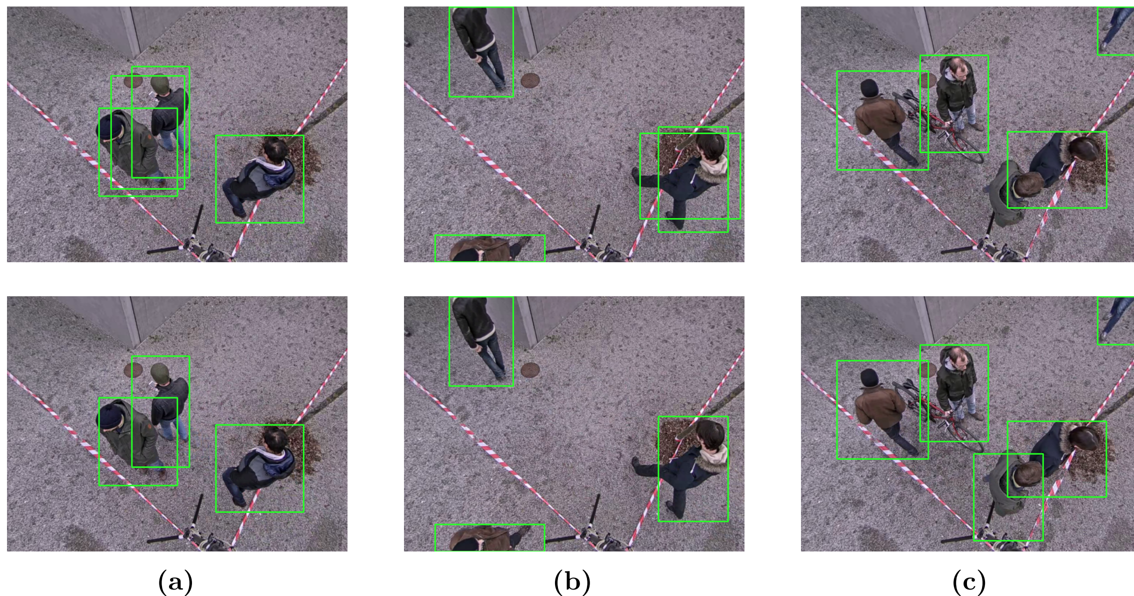
Figure 5.7: Precision *vs.* recall comparison of the best-performing NMS thresholds and Tnet for (a) the entire dataset and (b) images with at least one overlapping ground truth box.

Figure 5.7 additionally shows the corresponding precision *vs.* recall curves. While Tnet does not strictly outperform all NMS thresholds, it achieves significant higher recall levels

than the other models while still providing competitive precision. Further, in Figure 5.7 we see that the NMS threshold of greedy NMS is a trade-off between recall and precision. Low thresholds of 0.4 and 0.5 yield better precision than Tnet, but inferior recall. Higher threshold values, however, improve recall at the cost of lower precision. Tnet eliminates this drawback by providing a smooth and high enough precision over the entire recall range.

5.5.2 Qualitative Analysis

In this section, we will show some example images from the *Campus* dataset in Figure 5.8 and *Vienna* dataset in Figure 5.9 to qualitatively compare the detections of Tnet and the best-performing greedy NMS threshold. For a fair comparison, the detection thresholds are chosen such that both models operate on the same recall level (= 0.85).



(a) **(b)** **(c)**

Figure 5.8: Example images from the *Campus* dataset for qualitative comparison of the best-performing NMS model (top row) and Tnet (bottom row). Both models operate on the same recall level of 0.85.

The first example (a) from the *Campus* dataset shows a scene where greedy NMS fails to suppress the false positive in between the two true positives. While the hard-coded threshold is too large in this case, Tnet is able to predict all bounding boxes correctly. A similar example is illustrated in (b) where greedy NMS is not able to suppress a double-detection at the right border of the image.

Looking at the third sample (c), we see how Tnet is able to detect an additional person. Since the overlap of the detections is clearly smaller than 60 %, suppression is not caused by

NMS in this case. Instead, the confidence score is below the detection threshold. Although Tnet was not designed for such cases, it is able to increase (or decrease) confidence of detections.

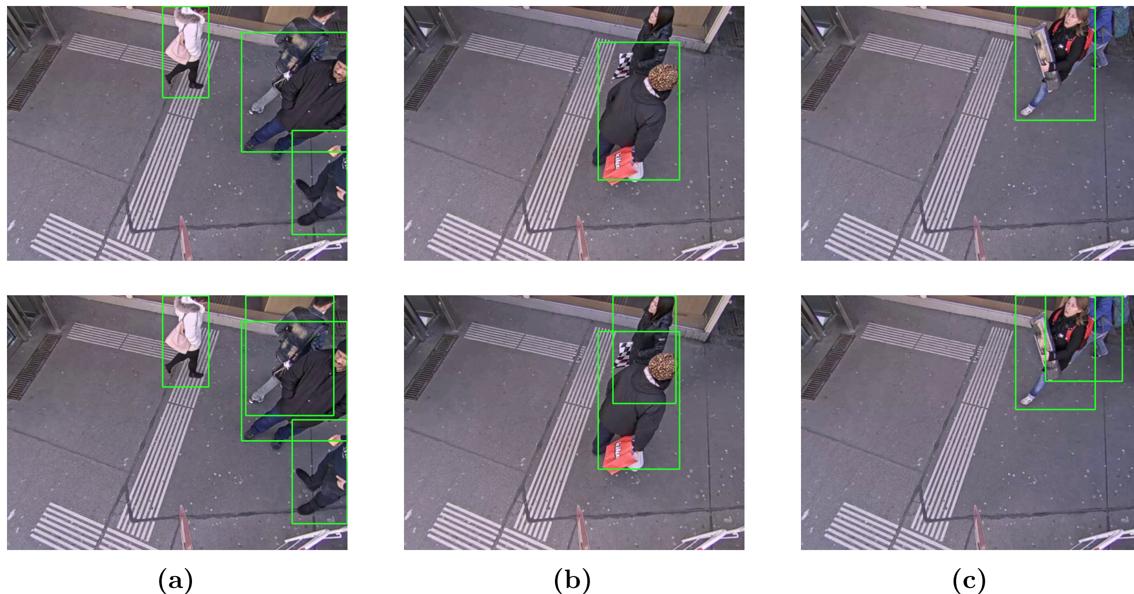


Figure 5.9: Example images from the *Vienna* dataset for qualitative comparison of the best-performing NMS model (top row) and Tnet (bottom row). Both models operate on the same recall level of 0.85.

The samples from Figure 5.9 focus on the improvement with crowded scenes and nearby standing people. All examples contain instances for which a NMS threshold of 0.6 is too low. A threshold of 0.7 or higher is necessary to keep the correct detections for these samples, though judging from Table 5.5 and Figure 5.7, this would lead to a significantly worse overall performance. However, Tnet is still able to choose the right detections in these cases.

5.6 Runtime Performance

Since the detections of our models are meant to serve as initializations of real-time tracking algorithms, runtime performance is crucial. In this section we analyze the runtime of our models during inference. All measurements were done with the hardware described in Section 5.3.2 running mainly on the GPU[1].

First, we measure the performance of the detection models without post-processing. The presented values are averaged with images of size 800×600 pixels. The baseline

[1]Some components of the models are implemented in Python™ and thus run on the CPU. All CNN operations are performed on the GPU using the CUDA® cuDNN backend of Caffe.

RGB-only model takes 0.067 s, the early-fusion model 0.087 s, and the late-fusion model 0.119 s per image. NMS costs ≈ 0.012 s resulting in the total runtimes of 0.079 s, 0.099 s, and 0.131 s for the models with greedy NMS.

The Tnet model used in our experiments approximately doubles the runtime of greedy NMS with 0.028 s. This results in a total runtime of 0.115 s per image for the early-fusion model combined with Tnet.

6 Conclusion and Outlook

Contents

6.1 Conclusion 73
6.2 Outlook and Future Work 74

In this work we investigated deep convolutional neural networks for the task of pedestrian detection in RGB-D images taken from an elevated viewpoint. In the following, we conclude the findings of this thesis and provide directions for potential future research.

6.1 Conclusion

Based on the work of Ren *et al.* [55], we showed how to fine-tune state-of-the-art CNNs for pedestrian detection in a specific scenario. In particular, we used models originally trained for image classification on general images to detect pedestrians from an overhead viewpoint. We also showed how to fine-tune these networks on a different image modality, namely depth data. We elaborated different approaches to fuse both modalities in a single network and showed how this combination improves the performance of the models and the quality of detections by performing evaluations on our custom datasets. Most noticeably, we found that the additional depth-modality improves the generalization capabilities of the detector compared to RGB images alone.

As most state-of-the-art models treat every detection independently (which results in multiple detections for the same object instance), incorrect detections need to be discarded in a post-processing step. The *de-facto* standard is to perform a greedy NMS to merge detections associated with the same object. This hand-crafted algorithm is a bottleneck for many detectors [35]. We implemented a learnable alternative proposed by Hosang *et al.* [34]

and incorporated it into our detection pipeline. The result is a single convolutional neural network which supports combined inference from images to detection boxes without the need of any post-processing in a single feed-forward pass. Further, the model eliminates the need of choosing a predefined overlap threshold which forces to trade-off precision and recall based on observations during validation. We showed that, unlike traditional greedy NMS, our model is able to adapt to regions with different pedestrian densities. Our experiments showed that the learned model achieves a performance boost especially for crowded scenarios, and the detection quality for less crowded scenes does not decrease.

6.2 Outlook and Future Work

Like all data-driven machine learning approaches, our detector would benefit from a larger set of training data. The current training set consists of samples from one scene with static background. Having samples with higher background variability would probably further increase the generalization capabilities of our models. Beside that, we expect that the performance gap between the different modality fusion approaches would become clearer.

We also assume that additional improvements could also be achieved by the availability of large-scale image datasets for depth images similar to the ImageNet [12] database for RGB images. Although we showed that it is possible to fine-tune RGB models for depth data, the 3-channel encoding is redundant and sub-optimal. Having general purpose depth models suitable for fine-tuning could further boost the performance of our models.

During the work on this thesis, other detection architectures competing with *Faster R-CNN* such as SSD [45] were proposed. An inherent difference of *SSD* to our models is the fact that it operates completely fully-convolutional and thus, the model directly outputs score maps which are needed by the NMS part of our proposed model. Having this direct output of score maps could pave the road for end-to-end training of the detection and NMS model. In this way, the models could benefit from each other during training which could possibly further improve the performance.

List of Acronyms

AP	*Average Precision*. 55, 59–61, 67
AuC	*Area under the Curve*. 55
BackProp	*Back Propagation*. 14, 16, 17, 19, 21, 22, 41, 49
CNN	*Convolutional Neural Network*. 6, 7, 14, 20, 36–38, 41, 43, 45, 47–49, 58, 59, 70, 71
DPM	*Deformable Part Model*. 5–7
FC	*Fully-connected*. 37–41, 43
FN	*False Negative*. 53, 54
FP	*False Positive*. 53, 54
GPU	*Graphics Processing Unit*. 22, 58, 70
HAG	*Height Above Ground*. 41, 59–61, 63
HoG	*Histogram of oriented Gradient*. 3, 5, 6, 25
IoU	*Intersection over Union*. 43–50, 55, 60
MLP	*Multi-layer Perceptron*. 14

NMS	*Non-Maximum Suppression*. 7, 8, 45, 47, 48, 50, 51, 53, 59, 66–72
NN	*Neural Network*. 14
PReLU	*Parametric Rectified Linear Unit*. 15, 16
RANSAC	*Random Sample Consensus*. 59
ReLU	*Rectified Linear Unit*. 15, 16, 23, 47–49
RoI	*Region of Interest*. 5–7, 35, 37–41, 43, 59, 60
RPN	*Region Proposal Network*. 36–44, 55, 59, 65, 66
SGD	*Stochastic Gradient Descent*. 18, 19, 28, 41, 59
SIFT	*Scale-Invariant Feature Transform*. 25
SVM	*Support Vector Machine*. 6, 33
TN	*True Negative*. 53, 54
Tnet	*Tyrolean network*. 45, 49–51, 58, 66–70
TP	*True Positive*. 53, 54

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. (page 17)

[2] Ahmed, I. and Carter, J. N. (2012). A robust person detector for overhead views. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*. (page 7)

[3] Alexandre, L. A. (2016). 3D Object Recognition Using Convolutional Neural Networks with Transfer Learning Between Input Channels. In *Proceedings of the International Conference on Intelligent Autonomous Systems*. (page 30)

[4] Azizpour, H. and Laptev, I. (2012). Object Detection Using Strongly-Supervised Deformable Part Models. In *Proceedings of the European Conference on Computer Vision (ECCV)*. (page 5)

[5] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. (page 17)

[6] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1 edition. (page 12, 16, 18, 19, 20, 21, 28)

[7] Bottou, L. (2012). Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer. (page 28)

[8] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. (2014). cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*. (page 58)

[9] Couprie, C., Farabet, C., Najman, L., and LeCun, Y. (2013). Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*. (page 30)

[10] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 5, 6, 26)

[11] deeplearning.net (2010). Convolutional Neural Networks (LeNet). `http://deeplearning.net/tutorial/lenet.html`. Accessed June 10, 2016. (page 23)

[12] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 23, 74)

[13] Dollár, P., Appel, R., Belongie, S. J., and Perona, P. (2014). Fast Feature Pyramids for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 36(8):1532–1545. (page 5)

[14] Dollár, P., Tu, Z., Perona, P., and Belongie, S. J. (2009). Integral Channel Features. In *Proceedings of the British Machine Vision Conference (BMVC)*. (page 6)

[15] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 26)

[16] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159. (page 19)

[17] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015). Multimodal Deep Learning for Robust RGB-D Object Recognition. In *Proceedings of the International Conference on Intelligent Robots and Systems*. (page 26, 31, 59)

[18] Everingham, M., Gool, L. J. V., Williams, C. K. I., Winn, J. M., and Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338. (page 5, 55, 58)

[19] Felzenszwalb, P. F., Girshick, R. B., McAllester, D. A., and Ramanan, D. (2010). Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9):1627–1645. (page 5)

[20] Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395. (page 60)

[21] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 4, 57)

[22] Girshick, R. (2015). Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*. (page 3, 6, 33, 34, 35, 42, 43, 59)

[23] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 3, 6, 26, 33, 34)

[24] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., and Bengio, Y. (2013). Maxout Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 28)

[25] Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. (page 31, 38, 39, 41, 56, 60)

[26] Gupta, S., Hoffman, J., and Malik, J. (2015). Cross Modal Distillation for Supervision Transfer. *arXiv preprint arXiv:1507.00448*. (page 32, 38)

[27] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition. (page 29, 30)

[28] He, K. and Sun, J. (2015). Convolutional Neural Networks at Constrained Time Cost. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 25)

[29] He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*. (page 35)

[30] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*. (page 25, 28)

[31] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the International Conference on Computer Vision (ICCV)*. (page 16, 28, 51)

[32] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*. (page 27)

[33] Hirschmüller, H. (2008). Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(2):328–341. (page 30)

[34] Hosang, J., Benenson, R., and Schiele, B. (2016a). A Convnet for Non-Maximum Suppression. In *Proceedings of the German Conference on Pattern Recognition (GCPR)*. (page 7, 45, 46, 49, 50, 51, 73)

[35] Hosang, J. H., Benenson, R., Dollár, P., and Schiele, B. (2016b). What Makes for Effective Detection Proposals? *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(4):814–830. (page 73)

[36] Hu, Q., Wang, P., Shen, C., van den Hengel, A., and Porikli, F. M. (2016). Pushing the Limits of Deep CNNs for Pedestrian Detection. *arXiv preprint arXiv:1603:04525*. (page 6)

[37] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160(1):106–154. (page 23)

[38] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*. (page 28, 43, 58)

[39] Karpathy, A. (2015). CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/`. Accessed June 19, 2016. (page 13, 21)

[40] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*. (page 19, 51)

[41] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. (page 16, 22, 23, 24, 28, 29, 43)

[42] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. (page 23)

[43] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer. (page 18, 29)

[44] Li, J., Liang, X., Shen, S., Xu, T., and Yan, S. (2015). Scale-aware Fast R-CNN for Pedestrian Detection. *arXiv preprint arXiv:1510:08160*. (page 6)

[45] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*. (page 6, 74)

[46] Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*. (page 26)

[47] Matan, O., Baird, H. S., Bromley, J., Burges, C. J. C., Denker, J. S., Jackel, L. D., LeCun, Y., Pednault, E. P. D., Satterfield, W., Stenard, C. E., and Thompson, T. J. (1992). Reading Handwritten Digits: A ZIP Code Recognition System. *IEEE Computer*, 25(7):59–63. (page 33)

[48] Murphy, K. P. (2012). *Machine learning: A Probabilistic Perspective*. MIT press, first edition. (page 10, 11, 12, 16)

[49] Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 16)

[50] Okutomi, M. and Kanade, T. (1993). A Multiple-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 15(4):353–363. (page 29)

[51] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training Recurrent Neural Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 51)

[52] Pinz, A. (2014). Bildgestützte Messverfahren. Lecture slides. (page 31)

[53] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12:145–151. (page 19)

[54] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *arXiv preprint arXiv:1506:02640*. (page 6)

[55] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. (page 3, 6, 7, 33, 34, 36, 37, 38, 43, 58, 59, 60, 73)

[56] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA. (page 19)

[57] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. (page 5)

[58] Santana, E., Dockendorf, K., and Principe, J. C. (2015). Learning joint features for color and depth images with Convolutional Neural Networks for object classification. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. (page 30)

[59] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013a). OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv preprint arXiv:1312.6229*. (page 6, 33)

[60] Sermanet, P., Kavukcuoglu, K., Chintala, S., and LeCun, Y. (2013b). Pedestrian Detection with Unsupervised Multi-stage Feature Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 33)

[61] Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 26)

[62] Shekhovtsov, A., Reinbacher, C., Graber, G., and Pock, T. (2016). Solving Dense Image Matching in Real-Time using Discrete-Continuous Optimization. In *Proceedings of the Computer Vision Winter Workshop (CVWW)*. (page 30, 56)

[63] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*. (page 22, 24, 25, 59)

[64] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net. *arXiv preprint arXiv:1412.6806*. (page 22)

[65] Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., and Babu, R. V. (2016). A Taxonomy of Deep Convolutional Neural Nets for Computer Vision. *arXiv preprint arXiv:1601.06615*. (page 20, 23, 24, 25, 26, 27, 28)

[66] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway Networks. *arXiv preprint arXiv:1505.00387*. (page 25)

[67] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 22, 25, 41)

[68] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, first edition. (page 29, 30)

[69] Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. (page 19)

[70] Uijlings, J. R., van de Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision (IJCV)*, 104(2):154–171. (page 34)

[71] van de Sande, K. E. A., Uijlings, J. R. R., Gevers, T., and Smeulders, A. W. M. (2011). Segmentation as selective search for object recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*. (page 6)

[72] Viola, P. A. and Jones, M. J. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 6)

[73] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of Neural Networks using DropConnect. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 28)

[74] Wang, S. I. and Manning, C. D. (2013). Fast dropout training. In *Proceedings of the International Conference on Machine Learning (ICML)*. (page 28)

[75] Xu, J., Ramos, S., Vázquez, D., and López, A. M. (2014). Domain Adaptation of Deformable Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 36(12):2367–2380. (page 6)

[76] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. (page 25)

[77] Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. (page 22, 24, 28, 43, 60)

[78] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning Deep Features for Scene Recognition using Places Database. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. (page 23)