Video-Based Environment Perception for Automated Driving using Deep Neural Networks

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für Maschinenbau
des Karlsruher Instituts für Technologie (KIT)

angenommene

DISSERTATION

von

M.Sc. Niels Ole Salscheider

Tag der mündlichen Prüfung: 07.07.2021

Hauptreferent: Prof. Dr.-Ing. Christoph Stiller
Korreferent: Prof. Mohan Trivedi

Zusammenfassung

Automatisierte Fahrzeuge benötigen eine hochgenaue Umfeldwahrnehmung, um sicher und komfortabel zu fahren. Gleichzeitig müssen die Perzeptionsalgorithmen mit der verfügbaren Rechenleistung die Echtzeitanforderungen der Anwendung erfüllen. Kamerabilder stellen eine sehr wichtige Informationsquelle für automatisierte Fahrzeuge dar. Sie beinhalten mehr Details als Daten von anderen Sensoren wie Lidar oder Radar und sind oft vergleichsweise günstig. Damit ist es möglich, ein automatisiertes Fahrzeug mit einem Surround-View Sensor-Setup auszustatten, ohne die Gesamtkosten zu stark zu erhöhen.

In dieser Arbeit präsentieren wir einen effizienten und genauen Ansatz zur videobasierten Umfeldwahrnehmung für automatisierte Fahrzeuge. Er basiert auf Deep Learning und löst die Probleme der Objekterkennung, Objektverfolgung und der semantischen Segmentierung von Kamerabildern. Wir schlagen zunächst eine schnelle CNN-Architektur zur gleichzeitigen Objekterkennung und semantischen Segmentierung vor. Diese Architektur ist skalierbar, so dass Genauigkeit leicht gegen Rechenzeit eingetauscht werden kann, indem ein einziger Skalierungsfaktor geändert wird. Wir modifizieren diese Architektur daraufhin, um Embedding-Vektoren für jedes erkannte Objekt vorherzusagen. Diese Embedding-Vektoren werden als Assoziationsmetrik bei der Objektverfolgung eingesetzt. Sie werden auch für einen neuartigen Algorithmus zur Non-Maximum Suppression eingesetzt, den wir FeatureNMS nennen. FeatureNMS kann in belebten Szenen, in denen die Annahmen des klassischen NMS-Algorithmus nicht zutreffen, einen höheren Recall erzielen.

Wir erweitern anschließend unsere CNN-Architektur für Einzelbilder zu einer Mehrbild-Architektur, welche zwei aufeinanderfolgende Videobilder als Eingabe entgegen nimmt. Die Mehrbild-Architektur schätzt den optischen Fluss zwischen beiden Videobildern innerhalb des künstlichen neuronalen Netzwerks. Dies ermöglicht es, einen Verschiebungsvektor zwischen den Videobildern für jedes detektierte Objekt zu schätzen. Diese Verschiebungsvektoren werden ebenfalls als Assoziationsmetrik bei der Objektverfolgung eingesetzt.

Zuletzt präsentieren wir einen einfachen Tracking-by-Detection-Ansatz, der wenig Rechenleistung erfordert. Er benötigt einen starken Objektdetektor und stützt sich auf die Embedding- und Verschiebungsvektoren, die von unserer CNN-Architektur geschätzt werden. Der hohe Recall des Objektdetektors führt zu einer häufigen Detektion der verfolgten Objekte. Unsere diskriminativen Assoziationsmetriken, die auf den Embedding- und Verschiebungsvektoren basieren, ermöglichen eine zuverlässige Zuordnung von neuen Detektionen zu bestehenden Tracks. Diese beiden Bestandteile erlauben es, ein einfaches Bewegungsmodell mit Annahme einer konstanten Geschwindigkeit und einem Kalman-Filter zu verwenden.

Die von uns vorgestellten Methoden zur videobasierten Umfeldwahrnehmung erreichen gute Resultate auf den herausfordernden Cityscapes- und BDD100K-Datensätzen. Gleichzeitig sind sie recheneffizient und können die Echtzeitanforderungen der Anwendung erfüllen. Wir verwenden die vorgeschlagene Architektur erfolgreich innerhalb des Wahrnehmungs-Moduls eines automatisierten Versuchsfahrzeugs. Hier hat sie sich in der Praxis bewähren können.

Abstract

Self-driving cars require a highly accurate environment perception to drive safely and comfortably. At the same time, the perception algorithms must fulfill the real-time requirements of this application with limited computational resources. Camera images provide a very important source of information for self-driving cars. They contain more details than the data from other sensors like lidar or radar. Camera sensors also tend to be cheap in comparison. This allows to equip a self-driving car with a surround-view sensor setup without increasing the total cost too much.

In this work, we present an efficient and accurate approach for video-based environment perception for self-driving cars. It relies on deep learning and solves the problems of object detection, object tracking, and semantic segmentation of the camera images. We first propose a fast CNN architecture for simultaneous object detection and semantic segmentation. This architecture is scalable so that accuracy can easily be traded for computation time by changing a single scaling factor. We then modify this architecture to predict embedding vectors for each detected object. These embedding vectors are used as an association metric when tracking the detected objects. They are also used for a novel Non-Maximum Suppression algorithm that we named FeatureNMS. FeatureNMS can achieve higher recall in crowded scenes where the assumptions of classical NMS fail.

We then extend our single-frame CNN architecture to a multi-frame architecture that takes two consecutive video frames as input. The multi-frame architecture predicts the optical flow between both frames inside the artificial neural network. This allows estimating a displacement vector between both video frames for each detected object. These displacement vectors are also used as an association metric when tracking the detected objects.

Finally, we present a simple and computationally cheap tracking-by-detection approach. It requires a strong object detector and relies on the embedding

and displacement vectors that are predicted by our CNN architecture. The high recall of the object detector results in frequent detections of the tracked objects. Our discriminative association metrics based on the embedding and displacement vectors allow for a reliable association of new object detections to existing tracks. These two components allow using a simple constant velocity motion model with a Kalman filter inside the tracking approach.

Our proposed methods for video-based environment perception show good results on the challenging Cityscapes and BDD100K datasets. At the same time, they are computationally efficient and can meet the real-time requirements of the application. We successfully employ our proposed architecture in the perception stack of a self-driving research vehicle. Here, it proved itself useful for real-world applications.

Preface

I conducted my research for this thesis during my time at the FZI Research Center for Information Technology in Karlsruhe. Here, I worked in the department ISPE-MPS that focuses on research in the context of self-driving cars. This research is done in close collaboration with the Institute of Measurement and Control Systems (MRT) at the Karlsruhe Institute of Technology. I would like to thank for all the support that I received from my employer which made this research possible.

Special thanks go to Prof. Christoph Stiller who supervised my thesis. He is also the director of ISPE-MPS and the head of MRT. In this function, he provided not only for the scientific environment but also for enough freedom to carry out this work successfully.

I would also like to thank Prof. Mohan Trivedi very much for being the co-examiner of my thesis and for hosting me at LISA × CVRR, UC San Diego. I spent around 3 months in his lab in 2019 and enjoyed my time there very much. It allowed me to concentrate on my research and the many discussions with the other Ph.D. students there gave me new ideas.

In this context, I would also like to thank the Karlsruhe House of Young Scientists (KHYS) which supported my stay at UCSD financially.

I am very thankful to all of my colleagues. I enjoyed working together with you and we had many productive scientific discussions. You did not only support me professionally but made working fun and became friends. I would like to thank Maximilian Naumann in particular who proofread a draft of my thesis.

Finally, I would like to express my thanks to my friends and family for the encouragement and moral support. You are always there for me.

Table of Contents

Acronyms and Symbols

Acronyms

ANN	Artificial Neural Network
AP	Average Precision
ASPP	Atrous Spatial Pyramid Pooling
AUC	Area Under Curve
BEV	Bird Eye View
BiFPN	Bidirectional Feature Pyramid Network
CNN	Convolutional Neural Network
ConvLSTM	Convolutional Long Short-Term Memory
CRF	Conditional Random Field
DPM	Deformable Part Model
EKF	Extended Kalman Filter
FCN	Fully Convolutional Network
FLOP	Floating Point Operation
FoV	Field of View
FPN	Feature Pyramid Network

FPPI	False Positives Per Image
GPGPU	General Purpose Computation on Graphics Processing Units
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HOG	Histograms of Oriented Gradients
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
KIT	Karlsruhe Institute of Technology
LAMR	Log-Average Miss Rate
LBP	Local Binary Patterns
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
mAP	Mean Average Precision
mIoU	mean Intersection over Union
MLP	Multi-Layer Perceptron
MOT	Multiple Object Tracking
MOTA	Multiple Object Tracking Accuracy
MOTP	Multiple Object Tracking Precision
MRT	Institute of Measurement and Control Systems
MSE	Mean Squared Error
MTL	Multi-Task Learning

NAS	Neural Architecture Search
NMS	Non-Maximum Suppression
PDF	Probability Density Function
ReLU	Rectified Linear Unit
RMS	Root Mean Square
RNN	Recurrent Neural Network
RPN	Region Proposal Network
SGD	Stochastic Gradient Decent
SVM	Support Vector Machine

Glossary

η	Learning rate
$\mathcal{L}$	Loss
$\mathcal{N}$	Normal distribution
$\mathcal{U}$	Uniform distribution

Notation in Neural Network Diagrams and Tables

Operator symbols

$\oplus$	Tensor concatenation along channel axis, i. e. stacking of feature maps.

+	Element-wise sum of feature maps.
·	Element-wise multiplication of feature maps.

Abbreviations for layers and modules

BN	Batch Normalization layer [IS15].
Conv	Convolution layer. The kernel size is usually given in round brackets.
DConv	Dilated convolution layer. The kernel size is usually given in round brackets.
DWConv	Depthwise separable convolution layer. The kernel size is usually given in round brackets.
FC	Fully-connected layer.
GAPool	Global average pooling layer.
LReLU	Leaky ReLU layer.
MBConv	Mobile inverted bottleneck module [San+18a].

Parameters

c	Number of channels / feature maps.
k	Kernel size of a convolution layer. "$k =$" can be omitted for brevity if the notation is unambiguous.
r	Dilation rate of dilated convolution layers.
s	Stride parameter, e. g. for convolution layers.

1 Introduction

Both the research community and the industry show strong interest in self-driving cars. Self-driving cars promise to deliver a safer and more comfortable driving experience. By communicating with other vehicles and the infrastructure, they can drive more anticipatory and help to save fuel. Driverless cars can ensure the mobility of people who are not able to drive otherwise, like elderly, children, or disabled persons. They open new business opportunities when used for taxi services and they reduce the need for parking spots close to popular destinations.

Many people hope that one day self-driving cars will reduce the number of traffic accidents significantly. For this, they need to surpass the abilities of human drivers. This includes their sensor ranges, perception and prediction accuracies, reaction times, and intelligent behavior planning.

These requirements are however difficult to fulfill. This opens a lot of interesting and exciting research questions and calls for constant improvement.

1.1 Motivation

One of the basic requirements of self-driving cars is a highly accurate environment perception. If the car does not know about the objects surrounding it, it cannot take them into account when planning. This would sooner or later lead to a collision.

Single-shot detections of surrounding objects are however not enough for self-driving cars. They might be sufficient for static infrastructure elements like traffic signs and traffic lights, but not for dynamic objects like other traffic participants. The reason is that the behavior of dynamic objects has to be predicted. Without a prediction, the planning module has no information about the future positions of dynamic objects and cannot take them into account. But the prediction module needs access to the past trajectories and behavior of

objects to make an informed prediction. Because of this, objects have to be tracked over time. This means that object detections from each time step are associated with past observations to form tracks.

In this thesis, we focus on camera images because they contain more details than the data from lidar or radar sensors. A lot of information is only available in camera images, like the status of traffic lights or the text on traffic signs. Cameras also have an advantage when it comes to the sensing range of some objects. Detecting an occluded pedestrian in the far distance, for example, is feasible using cameras, but very challenging when using other sensors. Visual information, on the other hand, is sufficient to handle most traffic scenarios. Human drivers demonstrate this every day. This makes vision-based sensors the most important and richest source of information about the environment. In practice, self-driving cars will fuse the information from multiple sensor types. There are situations when vision-based environment perception gives bad accuracy or fails, e. g. when there is fog or not enough light. The fusing of different information sources reduces the chance of failure since the different sensor types have different failure cases.

Camera images also contain information that is difficult to describe on an object level. An alternative is to classify each pixel of the image. We refer to this as pixel-wise semantic segmentation. It is, for example, suitable to extract the road surface. With this information, it is possible to verify that the planned trajectory lies in the drivable area. This information can also be used to extract the static parts of the scene like buildings and trees. These parts can then, for example, be matched to a map.

1.2 Contributions

The goal of this thesis is to develop a Convolutional Neural Network (CNN) architecture that is suitable for video-based environment perception for self-driving cars. It has to meet the run-time and accuracy requirements of the application while providing the required information. We target a frame rate of at least 10 Hz on an *Nvidia Quadro RTX 8000* Graphics Processing Unit (GPU). The presented approach provides pixel-wise semantic segmentation of the input camera images. It also provides single-frame object detections as well as tracked objects.

We verify the real-world applicability of the proposed approach in the research vehicle of the Institute of Measurement and Control Systems (MRT) at the Karlsruhe Institute of Technology (KIT). Here, it contributes to the perception stack of this self-driving car.

The main contributions of our work are the following:

- We present an efficient and scalable CNN architecture for simultaneous semantic segmentation and object detection based on EfficientDet [TPL20]. We employ a Multi-Task Learning (MTL) approach to efficiently obtain all predictions at once.
- We propose a new Non-Maximum Suppression (NMS) approach based on feature learning and integrate it in the CNN architecture.
- We integrate an optical flow estimation module based on PWC-Net [Sun+18] in this architecture. We use it to estimate a displacement vector between consecutive frames for each object.
- We present a simple and fast object tracking approach based on the information provided by the CNN architecture.

1.3 Outline

The remainder of this thesis is structured as follows:

Chapter 2 presents the technical background. It first introduces the basics of deep learning for image processing. It then presents related work in the areas of image classification, object detection, pixel-wise semantic segmentation, and object tracking.

In Chapter 3, we describe and evaluate our approach to simultaneous semantic segmentation and object detection as well as our new NMS approach. Both take single camera images as input and give single-shot predictions.

Chapter 4, on the other hand, is concerned with temporal information. We modify our approach from the previous chapter to work on pairs of camera images. This allows us to estimate the displacement of objects between frames.

We then introduce and evaluate a simple yet effective tracking approach based on the outputs of the neural network.

Chapter 5 finally concludes the thesis and gives an outlook.

2 Technical Background: Image Processing with CNNs

This chapter contains the technical background of our work. Section 2.1 shortly presents the history of classical object detectors and neural networks. After that, the basics of modern deep learning are introduced in Section 2.2. Section 2.3 and Section 2.4 contain related work in the areas of image processing and object tracking. In Section 2.5, we finally introduce the evaluation metrics for our approach.

2.1 From Classical Approaches to Artificial Neural Networks

Nowadays, image classification and object detection algorithms are mostly based on Artificial Neural Networks (ANNs). But this has not always been the case: Often, AlexNet [KSH12] is cited as the first hugely successful CNN architecture that brought the breakthrough of this technique in the year 2012 and made CNN based approaches highly popular.

Before, mainly classical methods were used for image classification, both in research and industry. These usually consist of a feature extraction and a classification stage. Object detection can be performed by applying an image classification algorithm in a sliding window approach: The classifier is evaluated on an exhaustive set of image crops. For each of these crops, the classifier decides if the crop is aligned with the outline of an object or not. The search space can be reduced by techniques like Selective Search [Uij+13] that use the low-level image structure to create proposals for the crops.

In the same way, image classification approaches were used to create pixel-wise semantic segmentation masks. A crop is generated for each pixel in the input

image, with this pixel in the center. Then, the task of the classifier is to classify the center pixel. The final segmentation mask can be generated by evaluating the classifier for each pixel of the image. Conditional Random Fields (CRFs) can be used to smooth the segmentation mask (e. g. [SCZ08]).

The features that are used in these approaches are mainly hand-crafted. Popular features include Local Binary Patterns (LBP) [OPH94], Haar features [VJ01], and Histograms of Oriented Gradients (HOG) [DT05]. These features are then used by classifiers to make the final prediction. Support Vector Machines (SVMs) [CV95] are one popular classifier. Decision trees are also widely used, especially in ensembles as random decision forests [Ho95].

Later object detection approaches like the Deformable Part Model (DPM) [Fel+10] or Regionlets [Wan+13] are based on the idea that deformable objects can be broken into rigid parts. The human body, for example, is deformable, but a human head is mostly rigid. The detector localizes the rigid parts of the object. If the parts are found and they fulfill certain relative positioning constraints, they provide evidence for the presence of the whole object.

Also ANN have been around for a long time. The perceptron [Ros57] forms their basis and dates back to 1957. In his original report, Rosenblatt designed the perceptron as an analog circuit with sensory input. Nowadays, the term is used for any binary classifier that calculates the following function:

$$f_{\mathbf{w},b}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Here, $\mathbf{x}$ is the vector of input values. The weight vector $\mathbf{w}$ and the bias b are parameters of the classifier. They have to be chosen using a suitable learning algorithm to solve the task at hand. The perceptron is a very simplified model of a biological neuron.

In 1969, Minsky and Papert showed that single-layer perceptrons are limited in what they can learn—it is, for example, impossible to model the XOR function with them since it is not linearly separable [MP69]. For this, Multi-Layer Perceptrons (MLPs) are necessary which consist of at least three layers: An input layer, a hidden layer, and an output layer. All layers except for the input layer contain neurons and all neurons use the values from the corresponding

previous layer as input. In a general MLP, each neuron performs the following calculation:

$$f_{\mathbf{w},b}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.2}$$

It is similar to the calculation performed by the perceptron (c. f. equation (2.1)) but the thresholding is replaced by a general nonlinear activation function $g(\cdot)$.

But in 1969, there was no efficient algorithm that could be used to train an MLP. After realizing this, many researchers lost interest in the field and research stagnated. This however changed in the year 1986, when the back-propagation algorithm was proposed [RHW86b; RHW86a]. It allows to efficiently calculate the gradient of the loss function (training objective) with respect to the weights (trainable parameters) of an ANN. Gradient descent can then be used to minimize the loss function and thus to train the neural network. Back-propagation is a special case of reverse mode automatic differentiation which forms the basis of modern deep learning frameworks.

Not much later, Cybenko presented the first version of the universal approximation theorem [Cyb89]. It states that continuous feed-forward neural networks with at least one hidden layer containing a sufficient number of neurons and sigmoid activation functions can approximate any arbitrary continuous function arbitrarily well. Other works (e. g. [Hor91]) show that the same holds true for many other activation functions.

At the same time, the first successful applications of ANNs were realized by training with the back-propagation algorithm. In 1989, LeCun et al. presented an ANN that could recognize handwritten zip codes from raw pixel values [LeC+89]. Their ANN already contains design patterns that form the basis of modern CNNs. It uses local filters with weight sharing to process the input feature maps of each layer and performs gradual downsampling.

Over the years, the available processing power increased and allowed for more complex ANNs that achieved higher accuracies. The performance of CNNs made a huge leap with the emergence of General Purpose Computation on Graphics Processing Units (GPGPU). This started the era of CNNs, with AlexNet [KSH12] being one of the first well-known networks. Since then, CNNs tend to outperform classical computer vision approaches.

2.2 Basics of Modern Deep Learning

This section covers the basics of modern deep learning. It introduces optimizers for neural networks in Section 2.2.1. Section 2.2.2 and Section 2.2.3 describe common building blocks of CNNs. Regularization strategies are introduced in Section 2.2.4 and approaches for the normalization of neuron activations are introduced in Section 2.2.5. Section 2.2.6 presents an overview of common loss functions to train CNNs. Finally, Section 2.2.7 introduces approaches for MTL.

2.2.1 Optimizers

In Section 2.1, we already introduced the MLP. It consists of multiple neurons that are arranged in layers. Most modern software frameworks and ANN architectures continue to use the notion of layers to structure the operations performed inside a network. The term *deep learning* emphasizes that these architectures can contain many layers stacked on top of each other, i. e. they are "deep". If a layer contains many neurons and thus produces many output values it is sometimes said to be "wide".

Current ANN architectures do not only contain fully-connected layers like the MLP. Instead, they consist of a variety of different operations which are combined depending on the task to solve. New layers are proposed as new problems are tackled, and current frameworks virtually set no limits to the operations used inside the networks.

Because of this, we can only describe a general ANN as a function $\Phi(\cdot;\theta)$:

$$\mathbf{y} = \Phi(\mathbf{x};\theta) \tag{2.3}$$

It calculates a (possibly multi-dimensional) output $\mathbf{y}$ given some (possibly multi-dimensional) input $\mathbf{x}$. Like the MLP, a general ANN has trainable parameters θ. The output of the model depends both on the values of the parameter vector θ as well as the computations performed by the model.

Both have to be chosen in a way that the ANN solves the task at hand. In practice, the model is designed first and then the parameter vector is optimized. The model design is done based on expert knowledge or using Neural Architecture Search (NAS). In both cases, it is an iterative process.

When designing, training, and evaluating a model, we need a way to judge its performance. A loss function $\mathcal{L}$ is used for this. A smaller loss value indicates that the model performs better. Therefore, the goal is to minimize the expected loss and thus achieve the best performance. Given a model Φ, we want to find the optimal parameter vector θ^* according to:

$$\theta^* = \arg\min_{\theta} \mathrm{E}\left(\mathcal{L}\left(\Phi(\mathbf{X};\theta), \mathbf{X}\right)\right) \tag{2.4}$$

Here, the random variable $\mathbf{X}$ is the input to the model. In practice, it is not possible to calculate the expected loss. It is instead approximated based on a finite set of data points $\mathbf{x}_i$. The parameter vector θ^* then minimizes the loss on these data points:

$$\theta^* = \arg\min_{\theta} \sum_i \mathcal{L}\left(\Phi(\mathbf{x}_i;\theta), \mathbf{x}_i\right) \tag{2.5}$$

Deep learning models can contain millions of parameters. Because of that, gradient-based first-order methods are usually employed to optimize them. The necessary gradients can be efficiently calculated using back-propagation [RHW86b; RHW86a] or more generally reverse-mode automatic differentiation. Here, we refrain from introducing the underlying mechanics of automatic differentiation. But the interested reader can find a good introduction to the topic in "Automatic Differentiation in Machine Learning: a Survey" [Bay+17].

Stochastic Gradient Descent

The simplest gradient-based optimization approach is vanilla gradient descent. In each iteration, the optimizer takes a step in the direction of the negative gradient:

$$\begin{aligned} \theta &\leftarrow \theta - \eta \cdot \nabla\mathcal{L} \\ \text{where} \quad \theta &= (\theta_1, \ldots, \theta_n)^T \\ \nabla\mathcal{L} &= \left(\frac{\partial\mathcal{L}}{\partial\theta_1}, \ldots, \frac{\partial\mathcal{L}}{\partial\theta_n}\right)^T \end{aligned} \tag{2.6}$$

The parameter η is the step size or *learning rate*. It is a hyperparameter that has to be tuned for each model. It is usually annealed during training so that the step size becomes smaller as training progresses. This helps convergence

as it prevents the optimizer to take too large steps, away from a (local) optimum close by.

Deep learning models do not only have many parameters—they also need enough training data points to get good estimates for them. It therefore usually is not possible to keep all training data points in the main memory of the computer. Because of this, the training is performed on mini-batches. In each training step, a new mini-batch is selected from the complete training set. Instead of calculating the loss $\mathcal{L}$ over all training examples, it is calculated over all examples in one mini-batch. The method is then referred to as Stochastic Gradient Decent (SGD).

One problem with SGD is that training can come to a halt if the optimization crosses a plateau or a small bump (local minimum) of the loss surface. It can also cause oscillations in valleys of the loss surface or when the gradient estimates are too noisy because of a small batch size. This problem can be solved by using an accumulated gradient to update the parameters, instead of the mini-batch gradient itself:

$$\begin{aligned} m &\leftarrow \mu \cdot m - \eta \cdot \nabla \mathcal{L} \\ \theta &\leftarrow \theta + m \end{aligned} \tag{2.7}$$

This method is called SGD with momentum. Here, μ is a hyperparameter that controls how quickly the old gradients are forgotten. A typical value is $\mu = 0.9$. The momentum m is initialized to 0 before the first iteration.

SGD with momentum is still widely used in research. There exist many other algorithms that converge more quickly and that are less sensitive to the right choice of hyperparameters. But ANNs trained with SGD tend to generalize well. Despite its simplicity, ANNs trained long enough with this algorithm can achieve better accuracy than when trained with more advanced algorithms.

Adam

Over time, many improvements to SGD have been proposed. Notable algorithms include AdaGrad [DHS10] and RMSProp [TH12]. AdaGrad divides the learning rate of each parameter by the Root Mean Square (RMS) of the accumulated previous gradients. This results in higher learning rates for parameters with

small gradients and the other way around. The intuition behind it is that this way, each parameter receives updates of similar magnitude. One problem is however that the training comes to a halt once the denominator containing the accumulated gradients becomes too large. RMSProp solves this problem by replacing the gradient accumulation of AdaGrad with an exponential moving average.

Nowadays, Adam [KB15] is a popular optimizer for ANNs. It takes ideas from both AdaGrad and RMSProp. When the right learning rate is used, it can train many models efficiently and achieve high accuracy. Adam performs the following calculations:

$$\begin{aligned} m &\leftarrow \beta_1 \cdot m + (1-\beta_1) \cdot \nabla \mathcal{L} \\ v &\leftarrow \beta_2 \cdot v + (1-\beta_2) \cdot (\nabla \mathcal{L})^2 \\ \hat{m} &= \frac{m}{(1-\beta_1)^t} \\ \hat{v} &= \frac{v}{(1-\beta_2)^t} \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{\hat{v}} + \varepsilon} \cdot \hat{m} \end{aligned} \tag{2.8}$$

The values m and v are exponential moving averages of the gradients and the squared gradients[1]. They are initialized to 0 before the first iteration. This causes a bias during the first iterations that has to be corrected. The bias corrected values for iteration t therefore are $\hat{m}$ and $\hat{v}$, respectively. The update of the parameter vector θ is then based on the momentum estimate, but scaled by the gradient RMS estimate like in AdaGrad and RMSProp. Typical values of the hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$. A good initial learning rate guess for the Adam optimizer is 10^{-3}.

Loshchilov and Hutter note that the Adam optimizer should be used with weight decay instead of ℓ^2 regularization (c. f. Section 2.2.4). They refer to this combination as AdamW [LH19].

[1] $(\nabla \mathcal{L})^2$ is computed element-wise

2.2.2 Common Layers in Convolutional Neural Networks

In the following, we will give a short overview of commonly used layers in CNNs for image processing. When dealing with images, these layers typically expect that the data is laid out as 3D tensors: Two dimensions are used for the spatial extension (along the y- and x-axis of the image), while the third dimension is used to store multiple values per position. The size of this 3D tensor is therefore given by the height H, width W, and the number of channels C. By slicing this 3D tensor along the last dimension, it can be broken into C individual feature maps of size $H \times W$.

In practice, multiple images are processed simultaneously in a mini-batch for efficiency reasons. When implementing a neural network, multiple of these 3D tensors are therefore stacked to obtain a 4D tensor. The first dimension then has the same size N (or B) as the mini-batch. The layer implementations, however, perform all computations independently on all examples in the mini-batch (with very few exceptions). We therefore ignore the batch dimension in the following paragraphs and only describe the computations that are performed on each individual sample.

While we are mostly interested in processing images and therefore operate on 3D tensors, the same principles also generalize to other dimensionalities.

Fully Connected and Convolutional Layers

The first MLPs solely consisted of fully connected layers. They are sometimes also referred to as dense layers. The reason for their name is that each neuron in the layer is connected to all possible inputs. Fully connected layers are rarely used for image processing nowadays: They contain many connections with associated trainable weights and thus require a lot of data to be trained.

Also, the number of input values is fixed for a fully connected layer once it is trained—adding or removing an input value would mean that weights have to be added or removed. This can, for example, be a problem when the ANN has to accept inputs of arbitrary size. Still, some image classification networks contain one or two fully connected layers close to the output.

The important correlations in a signal are often local. Imagine, for example, an image of a traffic scene. If the task is to decide if there is a pedestrian in the bottom left corner, then it makes sense to focus on this part of the image. It most likely does not help to search for cues in the top right corner of the image. This observation motivates to remove connections that would otherwise connect a neuron to an input that is spatially far away. In other words, it makes sense to connect each neuron only to inputs that are spatially close.

Another observation is that many tasks are translation invariant or translation equivariant. If the task from above changes to decide if there is a pedestrian in the bottom right corner, then it makes sense to focus on the corresponding part of the image. The local cues that indicate the presence of a pedestrian are however the same. This observation motivates to share weights between neurons at different locations. Two neurons that share weights would react to the same input patterns, just at different spatial locations of the signal.

Convolutional layers are based on these ideas of local connectivity and weight sharing. Their name comes from the relation to performing a convolution of the input signal with a filter map across spatial dimensions. Multiple neurons are placed at each discrete spatial location of the desired output feature maps. The number of neurons per location defines the number of output channels or output feature maps c. All neurons for one particular feature map share their weights. The local connectivity to the input feature maps is limited to a rectangle centered around the position of the neuron. The size of this rectangle is called the kernel size k.

It is often desired that the spatial resolution of the output feature maps matches the resolution of the input feature maps. This means, that neurons have to be placed at each spatial location of the input feature maps. If the kernel size is now larger than 1, the neurons at the border pixels have connections to spatial locations that lie outside of the input feature maps. In this case, padding has to be used to make the input feature maps (virtually) large enough. A popular choice is padding with zeros.

It is also often desired to only generate an output for every s-th input location. In this case, the convolution is performed (or the neurons are laid out) with stride s.

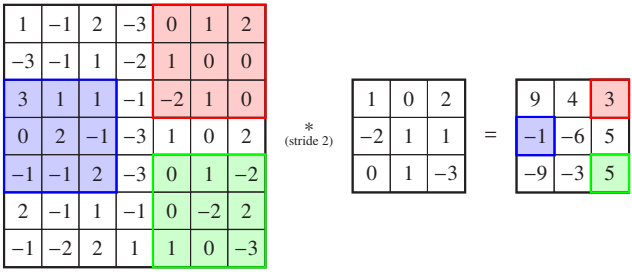


Figure 2.1: Example of a convolution on an input feature map with one channel. Here, the convolution has $c = 1$ output channels, a kernel size of $k = 3 \times 3$, a stride of $s = 2$ and no padding.

Figure 2.1 visualizes an example of a convolution on an input feature map with one channel. It produces one output feature map and has a kernel size of $k = 3 \times 3$, a stride of $s = 2$, and uses no padding.

In normal convolutional layers, a neuron is connected to all inputs covered by the kernel size. That means that it also accesses all feature maps (or channels) of the input. For each location in the output feature map, $k_x \cdot k_y \cdot c_{\text{in}} \cdot c_{\text{out}}$ multiplications and additions are performed. Here, the kernel size is $k_x \times k_y$ and there are c_{in} input feature maps as well as c_{out} output feature maps.

Depthwise separable convolutions are an alternative to normal convolutions and split the computation into two parts: First, separate convolutions are performed for each input feature map. These have a spatial extent (i. e. $k > 1 \times 1$), but only access one channel each (i. e. $c_{\text{in}} = 1$). This is followed by a convolution across feature maps without spatial extent (i. e. $k = 1 \times 1$) to fuse information across channels. In total, this results in $k_x \cdot k_y \cdot c_{\text{in}} + c_{\text{in}} \cdot c_{\text{out}}$ multiplications and additions per location. Even though not every convolution can be transformed into a depthwise separable convolution, they perform well for most applications and are more efficient in many cases. It should however be noted that the speedup by using depthwise separable convolutions is often lower than the difference in the number of multiplications and additions suggests. The reason is that depthwise separable convolutions are often memory-bound on GPUs, and not compute-bound.

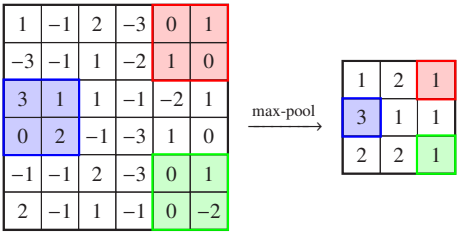


Figure 2.2: Example of a 2×2 max-pooling operation with a stride of $s = 2$.

Sometimes it is beneficial to have convolutions with a larger spatial extent to capture long-range correlations. This can be achieved by increasing the kernel size. But a larger kernel size means more computational overhead and more weights to learn. Dilated (or atrous) convolutions are a good alternative in this case. Here, the filter mask is dilated and the resulting gaps are filled with zeros. As a result, the filter mask covers a larger spatial extent but the number of weights stays the same.

Pooling Layers

Pooling layers aggregate information from multiple spatial locations. They can be used to make a CNN less sensitive, and therefore more robust, to translations in the input. This is desirable for e. g. image classification networks. But for other tasks like object localization or semantic segmentation, exact locations are important. Pooling layers can be detrimental in these applications.

Common pooling operations are max-pooling and average-pooling. They replace each input value by the maximum or average value within a surrounding window. Figure 2.2 visualizes an example of a 2×2 max-pooling operation with a stride of $s = 2$.

Resizing of Feature Maps

A common task in CNNs for image processing is upsampling and downsampling of feature maps. This is done to extract and fuse information at different levels

of detail. An alternative would be to keep all feature maps at a high resolution and only use dilated convolutions to capture long-range correlations. Doing so would however not be computationally efficient.

Both upsampling and downsampling can be performed by bilinear interpolation, possibly followed by a convolutional layer to reduce interpolation artifacts. An alternative for downsampling is to use either a convolutional layer or a pooling layer with a stride $s > 1$.

Upsampling can also be performed using transposed convolutions. They are also known as deconvolutions. In contrast to normal convolutions, transposed convolutions slide a filter mask across all locations of the output feature map, instead of across the input feature map. By using a stride $s > 1$, the spatial resolution of the output feature map is increased.

2.2.3 Activation Functions

It is necessary to apply nonlinear activation function between linear layers like convolutional or fully-connected layers. This is not only a requirement for the universal approximation theorem [Cyb89; Hor91], but also necessary to avoid that the linear operations collapse to one. This can be easily seen for fully-connected layers in the 1D case: They calculate a dot product between the weight matrix $\mathbf{W_i}$ and the input vector $\mathbf{x}_i$ as $\mathbf{y}_i = \mathbf{W}_i \cdot \mathbf{x}_i$. If the output of one layer is the input to a second layer, then the calculation simplifies to just another dot product:

$$\begin{aligned} \mathbf{y}_2 &= \mathbf{W}_2 \cdot \mathbf{x}_2 \\ &= \mathbf{W}_2 \cdot \mathbf{y}_1 \\ &= \mathbf{W}_2 \cdot (\mathbf{W}_1 \cdot \mathbf{x}_1) \\ &= (\mathbf{W}_2 \cdot \mathbf{W}_1) \cdot \mathbf{x}_1 \\ &= \tilde{\mathbf{W}}_1 \cdot \mathbf{x}_1 \\ \text{with } \tilde{\mathbf{W}}_1 &:= \mathbf{W}_2 \cdot \mathbf{W}_1 \end{aligned} \tag{2.9}$$

This collapsing is not possible if a nonlinear activation function is applied between the linear layers.

A lot of different activation functions have been proposed over the years. The most common ones in current CNN architectures are however the Rectified

Linear Unit (ReLU) and other variations based on it, the tanh and sigmoid functions, and the softmax function. These functions are defined as follows:

$$\begin{aligned} \text{relu}(x) &= \max(0, x) \\ \tanh(x) &= \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \\ \text{sigmoid}(x) &= \frac{1}{1 + \exp(-x)} \\ \text{softmax}\left((x_1, \ldots, x_n)^T\right) &= \left(\frac{\exp(x_1)}{\sum_{j=1}^{n} \exp(x_j)}, \ldots, \frac{\exp(x_n)}{\sum_{j=1}^{n} \exp(x_j)}\right)^T \end{aligned} \tag{2.10}$$

The ReLU activation function is commonly used throughout CNN architectures. It is very fast to compute since it requires just one comparison with zero. It achieves good performance for most tasks.

The tanh and sigmoid activation functions are commonly used in places where bounded values are required. The tanh activation function squashes the input value to the range from -1 to 1 and the sigmoid function squashes it to the range from 0 to 1.

The softmax function finally takes a real-valued input vector and normalizes it to a probability distribution. That means that all output values will be in the range from 0 to 1 and sum up to 1. It is frequently used as a last layer for classification problems when an estimate of class probabilities is desired.

One well-performing alternative to the ReLU activation function is the swish function [RZL17]. It was discovered using automatic search and tends to perform better than ReLU and its many other variations. It is defined as:

$$\text{swish}(x) = x \cdot \text{sigmoid}(x) = \frac{x}{1 + \exp(-x)} \tag{2.11}$$

Figure 2.3 provides a comparison between the ReLU and swish activation functions.

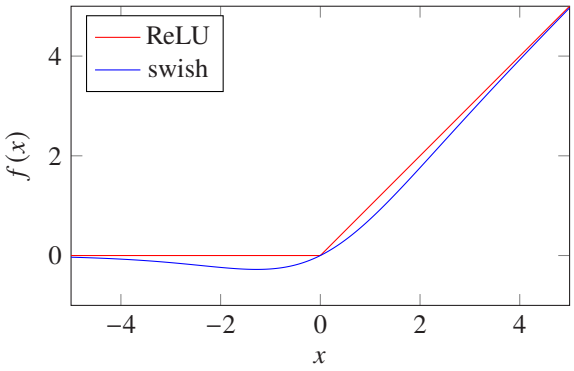


Figure 2.3: Comparison of the ReLU and the swish activation functions.

2.2.4 Regularization

Complex models containing many parameters are prone to overfitting when they are trained with too little data. That means that the model performs well on the training data, but not on unseen data. In other words, it just memorizes the training examples without generalizing.

Regularization techniques help to avoid overfitting. One popular approach is to add a penalty term for large weights $\mathbf{w}_i$ to the loss function. This helps to prevent that a (convolutional or fully-connected) layer focuses too much on single cues in the input data. We refer to this regularization approach as ℓ^1 or ℓ^2 regularization, depending on the utilized norm:

$$\begin{aligned} \mathcal{L}_{\ell^1} &= \mathcal{L}_{\text{task}} + \lambda \sum_i |\mathbf{w}_i| \\ \mathcal{L}_{\ell^2} &= \mathcal{L}_{\text{task}} + \lambda \sum_i \|\mathbf{w}_i\|_2 \end{aligned} \tag{2.12}$$

Weight decay is another technique that avoids large weights in the network. Here, the weights are decayed by a constant factor after each weight update by the optimizer. It is closely related to ℓ^2 regularization: If SGD is used for optimization, weight decay and ℓ^2 regularization are the same. This does

however not hold true in general. For the Adam optimizer, weight decay should be preferred over ℓ^2 regularization [LH19].

There are other regularization methods that aid model generalization. The dropout method randomly sets a fraction of the outputs of a layer to zero during training. It has the same effect as randomly removing some neurons from the neural network. This prevents that a neuron relies too much on a single input from another neuron.

Another well-known approach is data augmentation. The goal of it is to artificially increase the amount of training data. In the case of supervised learning, the training examples are changed in a way that preserves the desired labels. When working with image data, for example, the input image is often randomly cropped, blurred, or noise is added.

2.2.5 Normalization Strategies

Normalization layers are a key ingredient to train (very) deep neural networks. Without normalization, these networks are much more difficult and less efficient to train. The most popular normalization approach for CNNs is Batch Normalization [IS15]. The goal of Batch Normalization is to normalize each activation x to have zero mean and unit variance:

$$\hat{x} = \frac{x - \mathrm{E}(x)}{\sqrt{\mathrm{Var}(x) + \varepsilon}} \tag{2.13}$$

Here, ε is a small constant to avoid division by zero. In practice, $\mathrm{E}(x)$ and $\mathrm{Var}(x)$ are unknown and change with every training iteration. They are therefore replaced by (unbiased) estimates. During training, the estimates are calculated from the samples in the corresponding training batch. Moving averages are also maintained for these estimates. They are used during inference and makes the result independent of the other samples in the inference batch.

Ioffe and Szegedy argue that Batch Normalization is effective because it reduces the internal covariate shift [IS15]. The term internal covariate shift refers to the change of the distributions of network activations during training. Santurkar

et al. however claim that the true reason why Batch Normalization is effective is the significant smoothing of the optimization landscape [San+18b].

One problem with Batch Normalization is the dependence of the loss and the collected statistics on the mini-batch. This is especially an issue for distributed training on multiple GPUs or even machines: Either, the mini-batch statistics have to be synchronized between machines which can cause a high overhead. Or each machine performs the calculations only based on its own statistics. But this is more inaccurate because the estimates of the mini-batch statistics are noisier for smaller batch sizes. Another problem with Batch Normalization is that it does not always work well for Recurrent Neural Networks (RNNs). This is because the activations can have significantly different statistics during the recurring evaluations of the RNN.

Because of this, many alternatives to Batch Normalization have been proposed. These include, for example, Layer Normalization [BKH16], Instance Normalization [UVL17], and Group Normalization [WH18]. When used inside CNNs for image processing, they however often fail to reach the same accuracy as Batch Normalization. Recently, Online Normalization [Chi+19] has been proposed as another alternative. It does not rely on mini-batch statistics but uses exponentially decaying averages of the online samples. Because of this, it is suitable for small batch sizes and can even be used with a batch size of 1. The experiments of Chiley et al. show that ANNs with Online Normalization can achieve the same or even better accuracies as with Batch Normalization.

2.2.6 Loss Functions

As discussed in Section 2.2.1, ANNs are trained by minimizing a loss function. This section presents some of the most commonly used loss functions for learning classification tasks, regression tasks, and similarity metrics.

Classification

A classification task assigns a class label to each data point. In practice, ANNs often do not only output the most likely class, but a probability distribution over all classes. The most commonly used loss function for classification tasks

is the cross-entropy loss. The general multi-class formulation for one training example is as follows:

$$\mathcal{L}_{i,\text{CE}}(\mathbf{y}_i, \mathbf{p}_i) = -\sum_c y_{i,c} \cdot \log(p_{i,c}) \tag{2.14}$$

Here, $p_{i,c}$ is the predicted probability for class c of training example i. The ground truth label $y_{i,c}$ is 1 if the training example i has class c, and 0 otherwise. The individual values $p_{i,c}$ and $y_{i,c}$ form the elements of the vectors $\mathbf{p}_i$ and $\mathbf{y}_i$, respectively. The loss for multiple training examples is the sum over the individual losses.

Lin et al. propose Focal Loss [Lin+17c]. It can be computed as follows:

$$\mathcal{L}_{i,\text{Focal}}(\mathbf{y}_i, \mathbf{p}_i) = -\sum_c y_{i,c} \cdot \left(1 - p_{i,c}\right)^\gamma \cdot \log(p_{i,c}) \tag{2.15}$$

It is similar to the cross-entropy loss except for the modulating factor $\left(1 - p_{i,c}\right)^\gamma$. For $\gamma = 0$, it is identical to the cross-entropy loss. For $\gamma > 1$, it weights the easy examples down, so that the loss focuses on the hard examples. This is useful for highly imbalanced datasets with many easy examples and can make mining of hard examples superfluous. The authors find that $\gamma = 2$ works best in the context of object detection. This task has a high class imbalance between foreground and background examples.

Both cross-entropy loss and Focal Loss can be used in a weighted version: The loss per class can be multiplied by a class-specific weighting factor. This is also done to compensate for a class imbalance or to reduce the number of false negatives for a certain class.

Regression

Common loss functions for regression tasks are the ℓ^2 loss, the ℓ^1 loss, and the smooth ℓ^1 loss. The ℓ^2 loss of a predicted value $\tilde{y}_i$ for a true value y_i can be calculated as follows:

$$\mathcal{L}_{i,\ell^2}(y_i, \tilde{y}_i) = (y_i - \tilde{y}_i)^2 \tag{2.16}$$

The mean loss over all samples is also called Mean Squared Error (MSE). The ℓ^2 loss is often easy to optimize. If the error is large, also the gradient is large which helps a first-order optimizer to converge quickly. But it also means that this loss function is not robust to outliers since they can dominate the total loss.

The ℓ^1 loss, on the other hand, can be computed as follows:

$$\mathcal{L}_{i,\ell^1}(y_i, \tilde{y}_i) = |y_i - \tilde{y}_i| \tag{2.17}$$

The mean loss over all samples is also called Mean Absolute Error (MAE). It is not differentiable where the error is zero but has a constant gradient otherwise. This makes it more robust to outliers, but optimization is more difficult.

The smooth ℓ^1 loss, or Huber loss, combines both ℓ^1 and ℓ^2 losses as follows:

$$\mathcal{L}_{i,\text{Huber}}(y_i, \tilde{y}_i) = \begin{cases} \frac{1}{2} \cdot (y_i - \tilde{y}_i)^2 & \text{for } |y_i - \tilde{y}_i| \leq \delta \\ \delta \cdot |y_i - \tilde{y}_i| - \frac{1}{2} \cdot \delta^2 & \text{otherwise} \end{cases} \tag{2.18}$$

For small errors, the loss function behaves like the ℓ^2 loss and for larger errors it behaves like the ℓ^1 loss. This makes the Huber loss robust to outliers, but at the same time it has the desired properties of the ℓ^2 loss for normal data points. The threshold δ has to be chosen in a way that it separates outliers from normal data points.

Learning Similarity Metrics

Similarity metrics for images or image patches have a variety of applications. These include one-shot and few-shot learning [KZS15; Vin+16] as well as image retrieval, i. e. finding similar images based on a query image [Che+09; Wan+14; Son+16]. It can also be used to e. g. verify handwritten signatures [Bro+93] or faces [CHL05; SKP15]. There also has been a lot of work on object re-identification and tracking [Ber+16; He+18; DS18] based on similarity metrics.

For these tasks, the photometric similarity of image patches is not useful. Two images of two persons can be very similar on a raw pixel intensity level but still show different persons. Two images of the same person, on the other hand, can

have a very different appearance, if the lighting conditions, the background or the pose differ.

We are therefore interested in a similarity metric at higher levels of abstraction. It should be invariant to the viewpoint and the pose of the object, as well as to the illumination. It should also be robust in the presence of distortions like noise, blur, or dirt on the camera lens.

Since it is not obvious how to compare images at higher abstraction levels, it makes sense to learn a similarity metric. Most contemporary approaches are based on the concept of siamese neural networks. The term was coined in an early work by Bromley et al. [Bro+93]. The idea is the following: The two images that should be compared are processed by two CNN backbones. These two backbones share all trainable weights and are therefore identical copies. This explains the term "siamese neural network".

Another view is that both images are processed by the same backbone individually. The output of this backbone is a feature vector representation of the input. This representation captures the image contents at the relevant semantic level. The image similarity can then be calculated by the similarity of these feature vectors. Commonly used measures are the ℓ^2 distance or the cosine similarity.

The backbone is trained in a way that positive pairs (i. e. pairs belonging to the same class) have a small distance and negative pairs have a large distance. This requires a suitable training objective or loss function. One widely employed loss function is Contrastive Loss [CHL05; HCL06]. It is evaluated on image pairs using the following formula:

$$\begin{aligned} \mathcal{L}_{w,\text{contrastive}}(x_1, x_2, y_{1,2}) = & \frac{1}{2} \cdot y_{1,2} \cdot d_w^2(x_1, x_2) + \\ & \frac{1}{2} \cdot (1 - y_{1,2}) \cdot (\max(0, m - d_w(x_1, x_2)))^2 \\ \text{where} \quad d_w(x_1, x_2) = & \|g_w(x_1) - g_w(x_2)\|_2 \end{aligned} \tag{2.19}$$

Here, x_1 and x_2 form the pair of training images, and $y_{1,2}$ is the corresponding label. For positive pairs, $y_{1,2}$ has to be set to 1 and for negative pairs it has to be set to 0. The parameter m is the desired margin for negative pairs. The function $d_w(x_1, x_2)$ gives the distance of the feature vectors calculated by the siamese network: The backbone outputs the feature vector $g_w(x)$ given input x.

Here, the ℓ^2 norm is used to calculate the distance between the feature vectors. The final loss is then calculated by summing the losses of all training pairs in the current batch.

Triplet Loss [SKP15; Che+09] does not use pairs of training examples, but triplets. Each triplet consists of an anchor example x_a, a corresponding positive example x_p and a corresponding negative example x_n. The loss for a triplet can then be calculated as follows:

$$\begin{aligned} \mathcal{L}_{w,\text{triplet}}(x_a, x_p, x_n) &= \max\left(0, d_w^2(x_a, x_p) - d_w^2(x_a, x_n) + m\right) \\ \text{where} \quad d_w(x_1, x_2) &= \|g_w(x_1) - g_w(x_2)\|_2 \end{aligned} \tag{2.20}$$

The goal of this loss function is to ensure that the distance between the anchor and the negative example is at least by a margin m larger than the distance between the anchor and the positive example. Schroff, Kalenichenko, and Philbin also propose to use sampling strategy to select the triplets [SKP15]: Instead of training on all possible triplets, they select the hardest ones, i. e. the ones generating the highest loss.

Song et al. argue that a loss for metric learning should be calculated on all possible pairs of training examples in the current batch [Son+16]. In their work, they propose a loss function that can be efficiently calculated on all possible pairs of a training batch.

Manmatha et al. propose Margin based Loss instead [Man+17]. The loss for an image pair is calculated as follows:

$$\begin{aligned} \mathcal{L}_{w,\text{margin}}(x_1, x_2, y_{1,2}) &= \max\left(0, y_{1,2} \cdot (d_w(x_1, x_2) - b) + m\right) \\ \text{where} \quad d_w(x_1, x_2) &= \|g_w(x_1) - g_w(x_2)\|_2 \\ \text{s. t.} \quad \|g_w(x)\| &= 1 \quad \forall x \end{aligned} \tag{2.21}$$

Here, m again is the desired margin and b is the desired decision boundary. The label, however, has to be given as $y_{1,2} = 1$ for positive pairs and as $y_{1,2} = -1$ for negative pairs. In their experiments, Margin based Loss outperforms both Contrastive Loss and Triplet Loss. The authors also show that the sampling

strategy for negative training pairs is important for good results. They propose to sample a negative example x_2 given example x_1 with the following probability:

$$\Pr(x_2 = x|x_1) \propto \min\left(\lambda, q^{-1}(d_w(x_1, x))\right) \tag{2.22}$$

Here, λ clips the distribution and makes sure that all samples have at least a certain chance to be drawn. The Probability Density Function (PDF) $q(d)$ describes the probability to observe a pairwise distance d between points that are uniformly distributed on a unit hypersphere. This distribution approaches the normal distribution $\mathcal{N}\left(\sqrt{2}, \frac{1}{2n}\right)$ for large n, where n is the dimensionality of the feature space. This sampling approach gives better results than uniform sampling, hard negative mining and semi-hard negative mining.

2.2.7 Multi-Task Learning and Loss Weighting

The idea of MTL is to learn multiple tasks simultaneously using the same model. Two reasons make the use of MTL interesting: One is the reduction of computing time. If parts of an ANN are shared between multiple tasks, then the shared part has to be evaluated only once instead of multiple times. This can result in a significant speed-up when most of the network can be shared between several tasks.

The second reason why MTL can be beneficial is that it can result in improved accuracy of the model. Especially if the training datasets are small, a neural network is prone to overfitting. The different training objectives can act as a regularizer here, improving validation performance. The learned feature representations in an MTL setting tend to be more general because they have to carry useful information for multiple tasks. This often makes them more robust and allows to generalize better to unseen examples.

However, early work on MTL [Car97] already showed that it only works well when the tasks are related. Caruana provides an exhaustive study (using simple tasks and ANNs) about how the different tasks interact and affect each other, when MTL is effective, and much more. This work already recognizes that a too small network capacity hurts the performance of multi-task problems more than that of single-task problems. The increase in accuracy by MTL is therefore not only reached by reducing overfitting. This observation also shows

that forcing a shared feature representation for different tasks is not necessarily helpful—it is only the opportunity for sharing that can result in learning better features.

Recent work [Gon+19] confirms these findings. The authors report that MTL typically does not improve performance unless the tasks and loss weighting strategies are carefully selected.

Loss Weighting

The weighting of the different training objectives can be done in different ways. The naive way is to calculate a weighted sum of the objective functions to get the final objective function for optimization. The weights have to be tuned by hand to ensure that all tasks perform well. Usually, the individual terms are weighted to have approximately the same order of magnitude. This naive method is still used by many MTL approaches.

This approach has the disadvantage that it requires manual tuning of the weighting factors. But even worse, the learning of different tasks may proceed at a different speed. This makes the optimal weighting factors dependent on the training step.

Kendall, Gal, and Cipolla therefore propose an automated way to determine the weighting factors [KGC18]. The model outputs corresponding to the different tasks have different uncertainties. The authors assume that each output of the model is the desired output with some task-dependent additive Gaussian noise. They show that under this assumption, the maximisation of the log likelihood of the MTL model leads to the minimisation of the following loss term:

$$\mathcal{L} = \sum_i \frac{1}{2\sigma_i^2} \mathcal{L}_i + \log \sigma_i \tag{2.23}$$

Here, $\mathcal{L}_i$ are the individual task losses and σ_i^2 the corresponding variances. The authors now interpret all σ_i as trainable variables which are optimized by the optimizer like all other variables. This way, the weighting factors $\frac{1}{2\sigma_i^2}$ are estimated automatically. The additional penalty term $\log \sigma_i$ prevents the factors from becoming too small: The factor can only become small if σ_i becomes

large, but then the penalty term becomes large. This is important to avoid the trivial solution of the minimization where all factors become zero (assuming that all $\mathcal{L}_i$ are greater than zero).

In the experiments of Kendall, Gal, and Cipolla, the MTL models trained with their approach outperform the single-task models and the ones trained with hand-tuned weights.

Chen et al. propose another approach called GradNorm to solve the same problem [Che+18c]. The basic idea of their approach is to normalize the gradient magnitudes of the different objective functions in (some) shared layers of the network. To make the problem computationally less demanding, the authors suggest to perform this normalization only for the last layer that is shared between all tasks.

GradNorm first estimates the training progress of each task i. It uses the ratio $\frac{\mathcal{L}_i(t)}{\mathcal{L}_i(0)}$ of the current loss and the loss at the first training step for this. This ratio is normalized across all tasks to get a relative inverse training rate $r_i(t)$ for each task. The desired gradient magnitude in the last shared layer of the network is then defined as the average gradient magnitude times $[r_i(t)]^\alpha$. Here, α is a hyperparameter and controls how strongly a common training rate for all tasks should be enforced. The proposed GradNorm algorithm then optimizes the weighting factors of the individual losses to achieve the desired gradient magnitudes. In their experiments, the authors report better results for GradNorm then when using fixed weighting factors or the approach of Kendall, Gal, and Cipolla.

Inspired by GradNorm, Liu, Johns, and Davison propose Dynamic Weight Averaging [LJD19]. Like GradNorm, it weights the tasks based on the past changes of the corresponding losses. But unlike GradNorm, it does not normalize gradient magnitudes. Instead, it only tries to ensure approximately the same relative training progress for each task, compared to the corresponding initial losses.

Sener and Koltun argue that minimizing a weighted linear combination of per-task losses is only a valid approach if the tasks do not compete with each other [SK18]. They propose an optimization algorithm that can find a pareto optimal solution under realistic assumptions. In their experiments, this approach outperforms the ones by Kendall, Gal, and Cipolla and by Chen et al.

A recent publication [Gon+19] presents a comparison of different loss weighting strategies, including Dynamic Weight Averaging and the one by Kendall, Gal, and Cipolla. The authors conclude that a slightly modified version of the approach by Kendall, Gal, and Cipolla tends to work best, but there is no clear winner across all experiments.

Model Selection

MTL does not only require suitable training objectives and optimization algorithms, but also suitable models. A common approach across different domains is a shared backbone with task-specific network heads. Such an architecture requires that all tasks use the same input data, which is fed to the backbone. The backbone then extracts feature maps that contain relevant information for all tasks. These feature maps are fed into the task-specific network heads. Here, they are transformed to the final task-specific model output.

It is however not clear how much of the model should be shared between tasks, nor how much capacity the different model parts should have. This is usually determined empirically. Especially if the MTL problem contains many tasks, it can be difficult to find the optimal model.

Liu, Johns, and Davison propose to use an attention mechanism to select the relevant features for each task from the backbone output [LJD19]. Features that are irrelevant for the current task can be ignored, but still used for a different task where they are beneficial.

2.3 Common Image Processing Architectures

The previous section introduced the building blocks of CNNs and explained how the trainable parameters can be optimized. This section presents well-known architectures for different tasks. First, common backbone architectures for CNNs are described. Then, several task-specific network architectures for semantic segmentation, object detection, and optical flow estimation are introduced.

These architectures are usually developed based on a mixture of intuition, expert knowledge, and trial and error. NAS can automate this process to some degree by e. g. using genetic algorithms, reinforcement learning, or a grid search over design parameters.

2.3.1 Backbones for Image Classification

Nowadays, CNNs are used in nearly all areas of image processing. These include object detection and classification, pixel-wise semantic segmentation, depth and optical flow estimation, generative tasks, and much more. Over time, it became apparent that many of these tasks have common requirements. Certain building blocks and common network structures emerged and are now used in a wide variety of applications.

Most models that take one or more images as input first process them in a backbone. This backbone transforms the raw pixel values into a rich feature representation which is then used by task-specific network heads. This section gives an overview of common backbones that are widely used in today's CNN architectures. These backbones are usually developed for the task of image classification and are then transferred to other tasks.

One of the first CNN backbones that received high attention from researchers is AlexNet [KSH12], although it was not the first successful CNN architecture. In 2012, it outperformed the second-best approach in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Rus+15] by more than 10 percentage points. AlexNet seems simple compared to today's architectures, but at that time it was deeper than many other architectures. One of the insights of Krizhevsky, Sutskever, and Hinton was that the depth of this network helped it to achieve high accuracy. Training this comparably deep CNN was possible because of the advances made in GPGPU programming at this time. AlexNet consists of five convolutional and three fully-connected layers. It uses the ReLU activation function, max-pooling layers, and Local Response Normalization as feature normalization approach. Local Response Normalization normalizes the input feature maps in a local neighborhood, which results in a local contrast enhancement.

Another popular architecture that was published shortly after is VGG-16 [SZ14]. It is conceptionally similar and also consists of a series of convolutional and

max-pooling layers. These are followed by three fully-connected layers at the end of the network. With this network, Simonyan and Zisserman won the first place of the localization track of the ILSVRC 2014 [Rus+15]. This was possible by pushing the depth of the CNN to 16-19 layers. But doing so also increased the number of trainable parameters and the computational demands notably.

Inception Architectures

In the same year, GoogLeNet [Sze+15] won the classification track of the ILSVRC 2014. One of the main contributions of GoogLeNet is the introduction of the Inception module. Instead of stacking convolutional and max-pooling layers, the authors use a network-in-network [LCY13] approach to build modules. These modules are then stacked to form the complete CNN. Each module takes input feature maps, processes them with four parallel branches, and then concatenates their results to form the final output. These branches are

- a 1×1 convolution,
- a 1×1 convolution followed by a 3×3 convolution,
- a 1×1 convolution followed by a 5×5 convolution, and
- a 3×3 max-pooling operation followed by a 1×1 convolution.

Here, the 1×1 convolutions are used to reduce the number of input feature maps of the following operations. This also reduces the number of trainable parameters and the computational demands of the more complex 3×3 and 5×5 convolutions.

Not long after, Inception-v3 was published [Sze+16]. It introduces the use of different modules within one network. The authors replace the computationally expensive 5×5 and 7×7 convolutions of earlier architectures with less expensive ones: They recognized that an $n \times n$ convolution can be replaced by an $n \times 1$ convolution followed by a $1 \times n$ convolution while maintaining the size of the receptive field. An $n \times n$ convolution can also be replaced by a series of 3×3 convolutions. If this series contains enough 3×3 convolutions, then the same receptive field can be achieved. The authors show that this decreases the number of trainable parameters and the necessary number of computations under reasonable assumptions. Another improvement of Inception-v3 is the

use of Batch Normalization [IS15]. The presented insights guided the designs of many later architectures.

The Xception architecture takes the idea of saving computation time by replacing complex convolutions with multiple simpler ones one step further [Cho17]. Chollet interprets an Inception module with a maximum number of branches (or towers) as a depthwise separable convolution. Therefore, he proposes to replace the original Inception modules by depthwise separable convolutions. In his experiments, the Xception architecture outperforms Inception-v3.

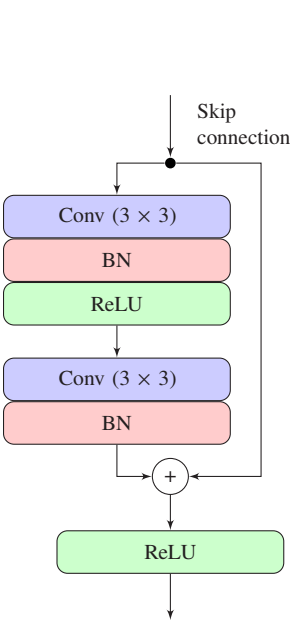
ResNet Architectures

Around the same time, the first ResNet architectures [He+16a] were published. Like Inception architectures, ResNet architectures consist of stacked modules. Each ResNet module learns a residual function with reference to the input. This can also be interpreted as each module decreasing the error in the output of the previous module.

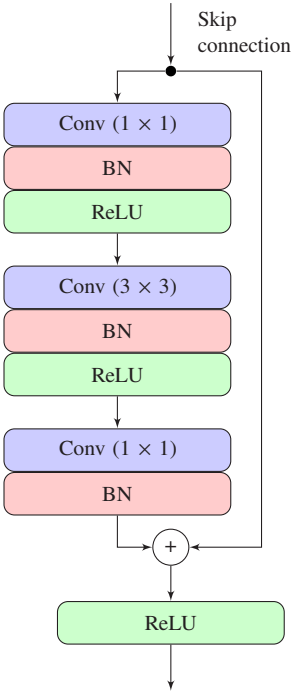
Let x be the input of a ResNet module, $h(x)$ the desired output, and $f(x)$ the corresponding residual. With this, we can write the mapping as $h(x) = x + f(x)$. The ResNet module calculates $h(x)$, but only the function $f(x)$ is learnt. This is achieved by adding a skip connection from the input to the output of the module. A visualization can be found in Figure 2.4.

This formulation is only possible if the input and output dimensions of the module match. Otherwise, a linear projection Wx of the input x is used in the skip connection to match the dimensions. In this case, the overall calculation changes to $h(x) = Wx + f(x)$.

He et al. show that their ResNet architecture allows to train very deep CNNs. In their experiments, they successfully train models with up to 1 202 layers. The design of the ResNet architecture is motivated by the observation that very deep VGG-style models have degraded accuracy compared to shallower ones. Empirically, training these models is difficult and the optimizer fails to find good optima. Theoretically, however, there exists a trivial solution: If some layers of a deep network compute an identity mapping, then these layers can be removed without changing the result to obtain a shallower network. This means that there is a set of weights with which the deeper network achieves the



(a) Normal ResNet module.



(b) Bottleneck ResNet module. It is used in ResNet-50 and bigger networks.

Figure 2.4: Structure of the original ResNet modules [He+16a]. The normal module in Figure 2.4a is used in smaller networks like ResNet-34. The bottleneck module in Figure 2.4b is used in larger networks like ResNet-50 and above. In the latter module, the computationally expensive 3×3 convolution is surrounded by two 1×1 convolutions. Their task is to reduce the number of input channels to the 3×3 convolution, and afterwards increase the number of output channels again. This helps to save computation time. For both the normal and the bottleneck module, it can happen that the number of input feature maps and desired output feature maps does not match. In this case, a projection is performed in the skip connection by placing an additional 1×1 convolution there.

same accuracy as the shallower ones. But it is difficult to discover this trivial solution during optimization. The skip connections of a ResNet module encode this knowledge explicitly.

After the original ResNet architecture was published, many improvements were proposed over time. In their follow-up work, He et al. propose to change the position of the activation functions and Batch Normalization layers inside the modules [He+16b]. The modified architecture achieves higher accuracy than the original one. Wu, Shen, and Hengel show that deeper networks are not always better, but that the right balance of network depth and width is key to good performance [WSH19]. Here, "width" refers to the number of feature maps per layer, and "depth" refers to the number of layers. The ResNeXt architecture [Xie+17] splits the residual branch of the module into multiple parallel branches while keeping the overall numbers of input and output channels constant. This is similar to how the Inception architecture is modified for Xception.

Szegedy et al. take inspiration from ResNet and propose Inception-ResNet architectures with skip connections [Sze+17]. This accelerates the training process considerably and can lead to better accuracy. The authors also present the Inception-v4 architecture without skip connections. Both achieve approximately the same accuracy.

MobileNet and EfficientNet Architectures

MobileNet [How+17] is an architecture that primarily focuses on efficient neural networks for mobile devices. It consists of stacked layers of depthwise separable convolutions, each followed by a Batch Normalization layer and a ReLU activation function. One key contribution of the MobileNet architecture is that it is scalable: It has a width and a resolution multiplier that can be used to adjust the number of channels in each layer as well as the input resolution. These hyperparameters allow tuning the model for either lower computation time or higher accuracy, depending on the application.

Howard et al. report that the proposed MobileNet variants have slightly lower accuracy than Inception-v3. They have however an order of magnitude fewer parameters and multiply-add operations. This makes the architecture suitable for applications with constrained computational resources.

MobileNetV2 [San+18a] continues to use depthwise separable convolutions. But the architecture is inspired by ResNet and consists of modules with skip connections. The width and resolution multipliers remain as tunable hyperparameters.

Howard et al. employ NAS to find improved architectures for MobileNetV3 [How+19]. This is based on and inspired by previous work [Yan+18; Tan+19; RZL17; ZL16; CZH18]. Changes from MobileNetV2 include the use of the swish activation function (or a computationally less expensive approximation thereof) and the "squeeze and excitation" technique [HSS18] in the residual branch.

Tan et al. also use NAS to find the MnasNet architecture [Tan+19]. The search space for NAS is designed based on the MobileNetV2 architecture.

Tan and Le systematically study the scaling of width, depth, and input resolution in CNN architectures [TL19]. They propose a "compound scaling" hyperparameter as a principled way to scale all three parameters at the same time. The goal is to achieve the best accuracy given the computational budget. Their empirical studies show that the ideal depth multiplier d, width multiplier w and resolution multiplier r can be calculated from the compound scaling parameter ϕ as follows:

$$\begin{aligned} d &= \alpha^\phi \\ w &= \beta^\phi \\ r &= \gamma^\phi \\ \text{subject to} \quad & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \quad \beta \geq 1, \quad \gamma \geq 1 \end{aligned} \tag{2.24}$$

The optimal values for α, β and γ have to be determined using a grid search.

Tan and Le first show the effectiveness of the proposed compound scaling parameter with ResNet and MobileNet architectures. Then they propose the scalable EfficientNet architecture. It was discovered by using NAS with similar settings as for MnasNet. The optimal values for the EfficientNet architecture are $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$. The scaling is done relative to the EfficientNet-B0 network that will be described in the following paragraphs.

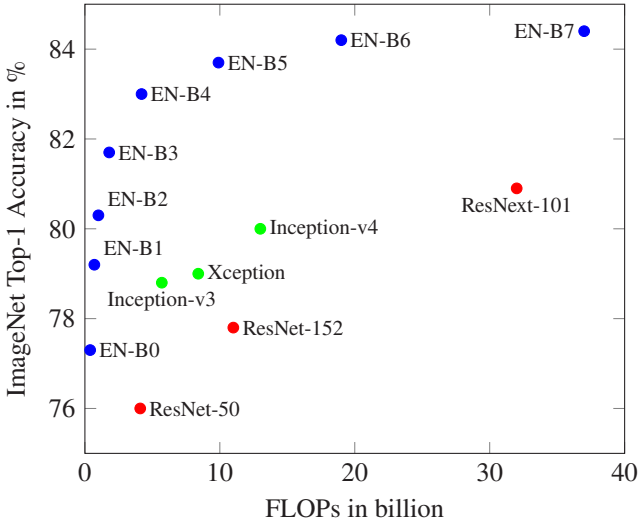


Figure 2.5: Accuracy on ImageNet [Rus+15] over FLOPs for different CNN backbones [TL19]. EN-B0 to EN-B7 is short for EfficientNet-B0 to EfficientNet-B7.

Figure 2.5 shows the accuracy over the number of Floating Point Operations (FLOPs) of some of the presented backbone architectures on ImageNet [Rus+15]. Being the latest step in the evolution of CNN backbones, the EfficientNet architecture achieves the highest performance for a given computational budget. The compound scaling hyperparameter allows generating efficient backbones for any use-case—be it with strong computational constraints or with high accuracy demands.

All EfficientNets use the mobile inverted bottleneck modules from MobileNet-V2 as the main building blocks. But similar to MobileNet-V3, EfficientNets use the swish activation function and the "squeeze and excitation" optimization [HSS18] inside their modules. Figure 2.6 shows the structure of the resulting module. Multiple modules are stacked to create an EfficientNet backbone. Table 2.1 describes the base EfficientNet-B0 architecture for image classification.

Using a skip connection in the mobile inverted bottleneck module is only possible if the input and output dimensions match. This is the case if the stride

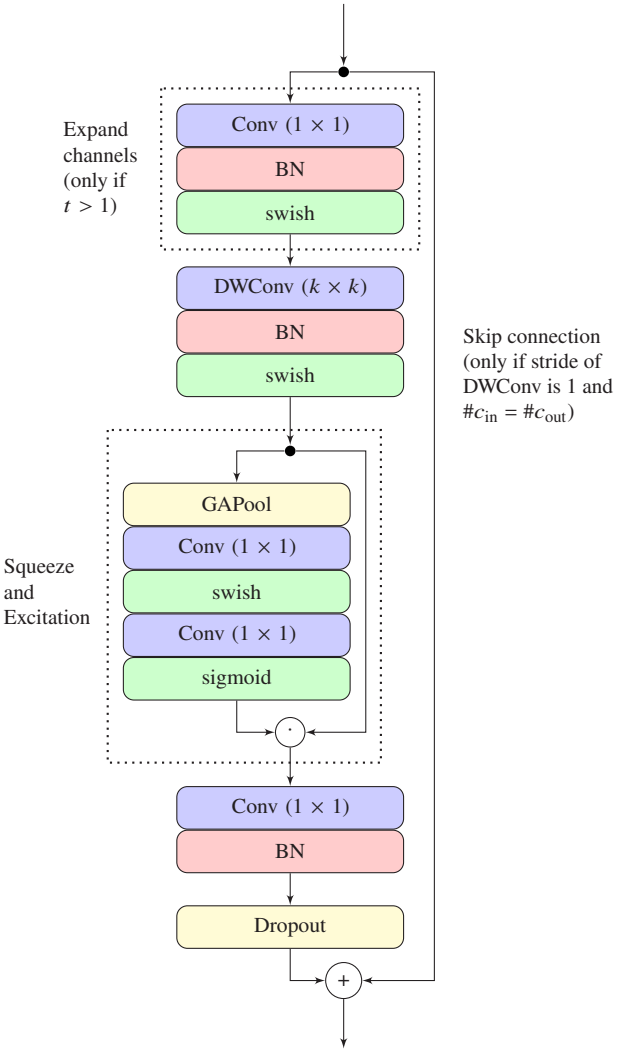


Figure 2.6: Structure of a mobile inverted bottleneck module as used in EfficientNets. The parameter t is the expansion factor from Table 2.1. We denote the number of input and output feature maps of the module as $\#c_{in}$ and $\#c_{out}$ respectively .

Operator	$h_i \times w_i$	c_i	n
Conv (3×3), BN, and swish	224×224	32	1
MBConv ($k = 3, t = 1$)	112×112	16	1
MBConv ($k = 3, t = 6$)	112×112	24	2
MBConv ($k = 5, t = 6$)	56×56	40	2
MBConv ($k = 3, t = 6$)	28×28	80	3
MBConv ($k = 5, t = 6$)	14×14	112	3
MBConv ($k = 5, t = 6$)	14×14	192	4
MBConv ($k = 3, t = 6$)	7×7	320	1
Conv (1×1), GAPool, and FC	7×7	1280	1

Table 2.1: The base EfficientNet-B0 architecture [TL19] for image classification. Each "MBConv" block is one instance of the mobile inverted bottleneck module from Figure 2.6. The parameters h_i and w_i are the height and width of the layer input, c_i is the number of input feature maps, and n indicates how often the block is repeated.

of all convolutions is 1 and the number of input and output channels is equal. Otherwise, the skip connection is omitted from the module.

The residual branch is divided into four parts: An expansion block, a depthwise separable convolution, a squeeze and excitation block, and a channel reduction block with a dropout layer. The expansion block expands the number of input channels by the expansion factor t. The 1×1 convolution inside this block can be viewed as a projection to a higher-dimensional space. The complete expansion block is omitted if the desired expansion factor t is not larger than one.

The following depthwise separable convolution is the only convolution with a filter size larger than 1×1. This convolution can be applied with a stride if the desired spatial resolution of the module's output is smaller than that of the input.

The depthwise separable convolution is followed by the squeeze and excitation block [HSS18]. This block first calculates a per-channel average activation value by using a global-average pooling operation. The resulting values are further

processed by the following convolution layers. The final sigmoid activation function then squashes the values to the range from 0 to 1. The result is used to modulate (weight) the input feature maps of this block.

The channel reduction block contains a final convolution. It reduces the number of channels to the desired number of output channels. It is followed by a dropout layer. This dropout layer is a regularization method and randomly drops neurons during training (c. f. Section 2.2.4).

2.3.2 Semantic Segmentation Architectures

The task of pixel-wise semantic segmentation is also a classification task. But in contrast to image classification, it assigns a class to each pixel of the input image. In the context of self-driving cars, a pixel could belong e. g. to a car, a pedestrian, the road surface, a sidewalk, and so on. The output of a pixel-wise semantic segmentation algorithm is again an image, where each pixel stores the corresponding class label.

One way to solve this task is to extract a patch for each pixel of the image by generating a crop with that pixel in the center. Each crop can then be classified independently to obtain the class of the center pixel. This approach has been used in early works [Cir+12], but it is very inefficient. Long, Shelhamer, and Darrell propose to use a Fully Convolutional Network (FCN) instead [LSD15]. They recognize that a fully-connected layer with one spatial output location can be replaced by a convolutional layer without changing the performed calculations. This interpretation, however, allows an efficient calculation for multiple overlapping patches, and therefore also for the complete image.

In the same publication, the authors fuse information from feature maps with different resolutions. The rationale behind it is to capture both global context from low-resolution feature maps and fine details from high-resolution feature maps. To fuse all feature maps, they rescale them to a common resolution and then sum them up. The upsampling of feature maps is performed with transposed convolutions.

Noh, Hong, and Han propose a nearly symmetrical encoder-decoder structure for semantic segmentation [NHH15]. First, features are extracted from the input image by a VGG-16 encoder. Here, the resolution of the feature maps is

successively decreased. The task of the decoder is then to upsample the feature maps back to the original resolution. The proposed decoder is a mirrored version of the encoder, where max-pooling layers are replaced by unpooling layers. Each unpooling layer forms a pair with a max-pooling layer. During downsampling, the max-pooling layer stores the indices of the selected values. The corresponding unpooling layer then copies its inputs to these locations and fills all other locations with zeros.

SegNet [BKC17] is also an encoder-decoder architecture that uses a VGG-16 backbone. It is conceptionally similar to the previous approach but fully convolutional.

UNet [RFB15] is an FCN and builds upon the ideas by Long, Shelhamer, and Darrell [LSD15]. It is an encoder-decoder architecture that uses max-pooling layers for downsampling and transposed convolutions for upsampling. It adds skip connections from the last feature map of each resolution in the encoder to the first feature map of the same resolution in the decoder.

Also the first version of DeepLab [Che+14] uses a VGG-16 backbone. But in contrast to the previously presented approaches, this architecture only computes feature maps of higher resolutions. The authors remove the last max-pooling layers of the VGG-16 backbone to achieve this. All following convolutions are then replaced by dilated convolutions to obtain the same receptive field. With these modifications, the feature maps are downsampled by a factor of 8 in the backbone. The original image resolution is then restored by using bilinear upsampling. Finally, the network output is refined using a CRF.

Chen et al. evaluate different architectures for the second version of DeepLab [Che+18a]. They replace the VGG-16 backbone with a more modern ResNet-101 backbone. One of the key contributions is the use of Atrous Spatial Pyramid Pooling (ASPP) modules inside the network. Inside an ASPP module, the input feature maps are transformed by four parallel branches and then summed up. Each branch contains a dilated convolution layer with a different dilation rate. The ASPP module allows capturing details at different scales. It improves the reported accuracies significantly.

DeepLabv3 [Che+17] still uses a ResNet backbone with an ASPP module at the end. The authors use dilated convolutions inside the backbone and modify the ASPP module slightly. It now contains 5 branches: A 1×1 convolution, three 3×3 dilated convolutions with different dilation rates and global-average

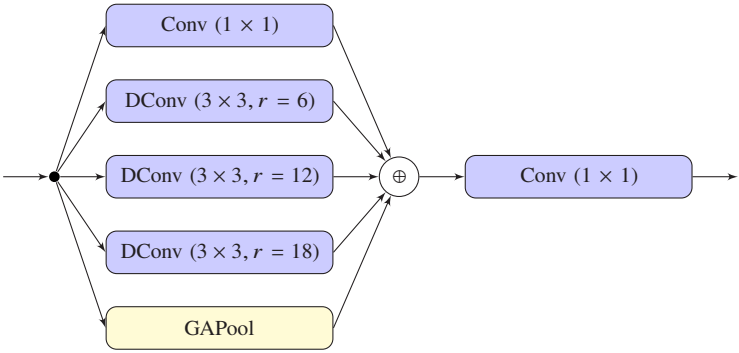


Figure 2.7: ASPP module as proposed by Chen et al. [Che+17].

pooling operation. The outputs of these branches are then concatenated. A final 1×1 convolution reduces the number of feature maps. A visualization of this module can be found in Figure 2.7.

The pyramid pooling module of PSPNet [Zha+17] follows a similar approach as ASPP. But instead of using dilated convolutions, it performs the following steps to capture details at different scales: First, the module performs multiple downsampling operations on the input feature maps to get feature maps at different resolutions. These are then independently processed by convolutional layers. After that, they are upsampled to the original resolution. Finally, all resulting feature maps are concatenated.

DeepLabv3+ [Che+18b] combines the ASPP module from earlier DeepLab versions with an encoder-decoder structure that is inspired by UNet and SegNet. The ASPP module is placed between the encoder and the decoder, where the feature maps have the lowest resolution. Figure 2.8 depicts the high-level structure of the architecture.

RefineNet [Lin+17a] is an encoder-decoder architecture with a ResNet backbone. It restores the original resolution in the decoder by successively fusing upsampled feature maps with earlier feature maps of the same resolution. The high-level structure resembles UNet, but the fusion approach is more advanced.

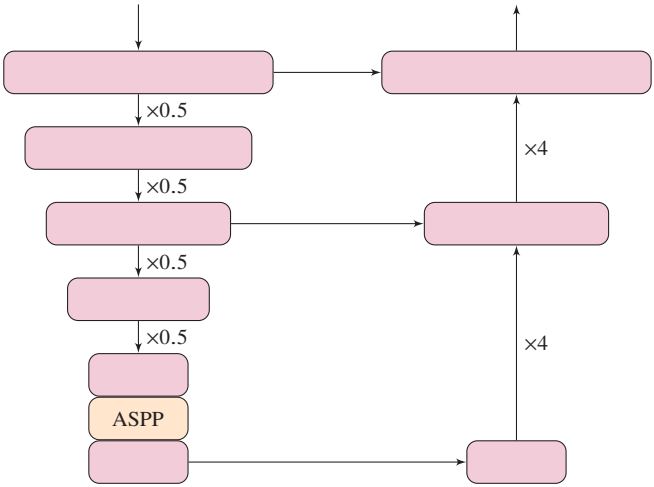


Figure 2.8: High-level structure of DeepLabv3+ [Che+18b]. Each red block can contain multiple ResNet modules and other layers. The width of the block is used to visualize changes in feature map resolution because of downsampling and upsampling.

Recently, Panoptic Feature Pyramid Networks [Kir+19] were published. Kirillov et al. add a semantic segmentation head to a Feature Pyramid Network (FPN) and solve the instance segmentation and semantic segmentation tasks simultaneously. The FPN fuses feature maps of different resolutions to capture details at different scales. FPNs are commonly used for object detection architectures and will be presented in Section 2.3.3. The output feature maps of the FPN are fused again, and then fed to an object detection and a mask prediction network head. This idea is similar to our approach that will be presented in Section 3.1. The authors report a similar accuracy as DeepLabv3+ (79.1 % vs. 79.6 %) on the Cityscapes validation dataset [Cor+16].

Table 2.2 lists the reported mean Intersection over Union (mIoU) on the semantic segmentation task of the PASCAL VOC 2012 test dataset [Eve+12] for some of the presented approaches. The key insights of the presented comparison are the following:

Approach	mIoU
DeepLabv3+ [Che+18b]	87.8 %
DeepLabv3 [Che+17]	85.7 %
PSPNet [Zha+17]	85.4 %
DeepLabv2 [Che+18a]	79.7 %
DeepLabv1 [Che+18a]	66.4 % - 70.3 %
FCN-8s [Che+14]	62.2 %

Table 2.2: Reported semantic segmentation results of different architectures on the PASCAL VOC 2012 test dataset [Eve+12].

- Most semantic segmentation architectures reuse backbones developed for image classification.
- There are different architectures with significantly different design ideas that achieve good performance.
- Most architectures are based on an encoder-decoder architecture. The feature maps are downsampled in the encoder and upsampled in the decoder. Early downsampling is computationally more efficient.
- The fusion of feature maps at different resolutions is important for good accuracy. Otherwise, finer details get lost due to the downsampling operations. There are different fusion approaches that use e. g. skip connections, ASPP modules, pyramid pooling modules, or an FPN.

2.3.3 Object Detection Architectures

The object detection task can be broken into an object classification and an object localization task. It can be solved naively with an image classification network: By using the sliding window approach of classical object detection, an exhaustive list of potential object bounding boxes can be generated. Each bounding box in this list can then be classified as containing an object of interest or not.

This naive approach is computationally very expensive and is not used in practice. But it shows the connection between image classification and object detection, and motivates why the backbones presented in Section 2.3.1 are also a good choice for this task. Like the semantic segmentation architectures presented in the previous section, most contemporary object detection networks are CNNs architectures.

Existing object detection architectures can be divided into single-stage and two-stage detectors. Single-stage detectors process the input image with a backbone and then directly output object detections in the network heads. Two-stage detectors, on the other hand, first generate object proposals. The second stage then refines these proposals and classifies them as containing an object of interest or not. Two-stage detectors tend to have slightly higher accuracy but are also computationally more demanding.

Two-Stage Detectors

The most popular two-stage detectors are part of the R-CNN family. The original R-CNN architecture was proposed by Girshick et al. [Gir+14]. First, Selective Search [Uij+13] is used to extract region proposals from the input image. These region proposals form the candidates for the detected objects, but not all proposals correspond to objects. A crop of the image is generated for each region proposal. It is then resized to a fixed size and fed to a CNN. This CNN, which was trained on a general image classification task, extracts features of the corresponding image patch. Then, a linear SVM [CV95] classifies each proposal based on the extracted features. The result is either an object class or the background class, meaning that the proposal does not contain an object. Finally, the localization of objects is improved by regressing a bounding box relative to the region proposal. This is also done based on the CNN features. Duplicate detections are suppressed by NMS.

The following formulas are used to calculate the regression targets t_x, t_y, t_w, and t_h for the bounding box refinement. In the first version of R-CNN, the regression is relative to the region proposals. The same formulas are used in later versions of R-CNN and many other object detection architectures. But there, they are used to regress bounding boxes relative to anchor boxes. The tuple (x, y, w, h) describes the bounding box of an object by the center coordinates x and y as

well as the width w and the height h. The tuple (x_a, y_a, w_a, h_a) describes the corresponding anchor bounding box or region proposal in the same way.

$$\begin{aligned} t_x &= \frac{x - x_a}{w_a} \\ t_y &= \frac{y - y_a}{h_a} \\ t_w &= \log\left(\frac{w}{w_a}\right) \\ t_h &= \log\left(\frac{h}{h_a}\right) \end{aligned} \tag{2.25}$$

It follows that the bounding box $(\tilde{x}, \tilde{y}, \tilde{w}, \tilde{h})$ of a detected object can be calculated from the network output $(\tilde{t}_x, \tilde{t}_y, \tilde{t}_w, \tilde{t}_h)$ and the corresponding anchor location (x_a, y_a, w_a, h_a) and as follows:

$$\begin{aligned} \tilde{x} &= \tilde{t}_x \cdot w_a + x_a \\ \tilde{y} &= \tilde{t}_y \cdot h_a + y_a \\ \tilde{w} &= w_a \cdot \exp(\tilde{t}_w) \\ \tilde{h} &= h_a \cdot \exp(\tilde{t}_h) \end{aligned} \tag{2.26}$$

The original R-CNN architecture is very slow because the CNN has to be evaluated for every region proposal. Girshick reports that inference takes 47 s per image on a GPU [Gir15]. Fast R-CNN [Gir15] improves on that by running a CNN only once on the whole image to obtain a feature map. The region proposals are then used to crop the corresponding part of the feature map, and not of the input image. A max-pooling operation ensures that the representation of each crop has a fixed size. The author refers to this as RoI pooling. A network head then classifies each of the resulting proposal representations and performs a bounding box regression. Fast R-CNN is based on a VGG-16 backbone. It is pretrained on an image classification task but then fine-tuned for the object detection task. End-to-end training by back-propagation through the RoI pooling layer is possible after assigning region proposals to ground truth objects during training. The assignment is based on the Intersection over Union (IoU) between the region proposal and the ground truth box.

Ren et al. propose the Faster R-CNN architecture [Ren+15] which is faster than Fast R-CNN. It replaces Selective Search with a Region Proposal Network (RPN). The input image is first processed by a backbone and the RPN to generate region proposals. In Faster R-CNN, the region proposals are based on anchor boxes. These anchor boxes are prototype boxes with a certain size and aspect ratio. The set of all anchor boxes is generated by placing prototype boxes at all possible discrete locations in a sliding window fashion. The RPN then predicts an objectness score for each anchor. This objectness score indicates how likely it is that the anchor contains an object (i. e. that it does not belong to the background class). The generated region proposals are then processed in the same way as in the Fast R-CNN architecture.

Mask R-CNN [He+17] adapts Faster R-CNN to the instance segmentation task. This means that it generates binary instance masks for all object instances. The overall structure is nearly identical with Faster R-CNN, but an additional branch is added to the object detection head. This branch consists of a series of convolutions and predicts a pixel-wise object mask.

Lin et al. integrate an FPN into the Faster R-CNN architecture to fuse information at different scales [Lin+17b]. The CNN backbone of Faster R-CNN produces feature maps at different resolutions. Earlier layers produce feature maps with higher resolutions, while later layers produce feature maps with lower resolutions due to downsampling. Similar to an image pyramid, these feature maps form a feature pyramid and capture details at different scales. The FPN is constructed by gradually upsampling smaller feature maps, and by adding skip connections between feature maps with the same resolution. This is visualized in Figure 2.9. The network heads to generate region proposals and to detect objects are then attached to all pyramid levels.

Single-Stage Detectors

In contrast to two-stage object detectors, single-stage detectors do not rely on region proposals. Instead, they directly perform object detection on the input image. This approach tends to be simpler and computationally less expensive.

YOLO [Red+16] is an early work that focuses on real-time object detection. The authors report that it is 2.5 to 6 times faster than Faster R-CNN, depending on the network backbone. The YOLO architecture consists of a single CNN

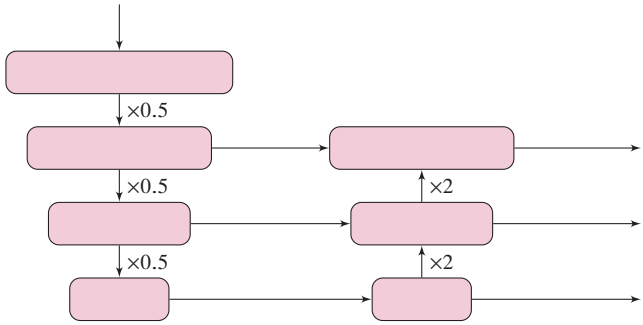


Figure 2.9: Conceptual structure of the FPN proposed by Lin et al. [Lin+17b]. The width of each block is used to indicate the feature map resolution. Different blocks have different resolutions because of downsampling and upsampling operations in the network. The high-resolution feature maps of earlier layers are fused with upsampled feature maps of later layers to restore fine details.

backbone, followed by two fully-connected layers. It divides the image into cells and predicts a fixed number of bounding boxes per cell. Each prediction contains a regression of the box position relative to the cell center, a regression of the box size relative to the image size, and a confidence score. A cell is responsible to detect an object if the center of the object is inside this cell.

YOLOv2 [RF17] is an improved version of YOLO. The authors remove the fully-connected layers and make the network fully convolutional. The image is not divided into cells anymore. Instead, the network relies on anchor boxes similar to Faster R-CNN. YOLOv3 [RF18] is the third iteration of YOLO. It adds many smaller improvements to the YOLOv2 architecture and training process.

The SSD architecture [Liu+16] was published after the first version of YOLO but before the second. It uses a VGG-16 backbone and is also based on anchor boxes. The authors recognize that predictions at different scales are important for high recall. For this, they employ successive downsampling of the backbone output to obtain feature maps at different resolutions. All resulting feature maps are then fed to a network head which predicts the object classes as well as the box regression parameters relative to the corresponding anchor boxes.

RetinaNet [Lin+17c] is based on a very similar anchor box design. It however uses a ResNet backbone together with the FPN design by Lin et al. [Lin+17b]. Most previous architectures solved the class imbalance between the foreground and background class by employing a sampling strategy. RetinaNet uses Focal Loss (c. f. Section 2.2.6) instead. This loss down-weights easy examples so that they cannot dominate the loss and suppress more informative examples. Lin et al. report that RetinaNet outperforms Faster R-CNN based models, YOLOv2, and SSD on the COCO "test-dev" dataset [Lin+14].

Like Mask R-CNN, PANet [Liu+18] solves the instance segmentation task. PANet uses a ResNet based backbone followed by a modified FPN structure. This modified FPN has a second augmentation path which enriches the information in the low-resolution feature maps. PANet fuses the output of the FPN again and feeds the result to an object detection and a mask prediction network head.

EfficientDet [TPL20] is based on the EfficientNet backbone. Like the backbone, EfficientDet can be scaled to meet the performance requirements and computational constraints of the application. It reuses the depth d and width w scaling parameters from EfficientNet (c. f. equation (2.24)). It introduces the following additional formulas for scaling based on the compound scaling factor ϕ:

$$\begin{aligned} d_{\text{bifpn}} &= 3 + \phi \\ d_{\text{head}} &= 3 + \left\lfloor \frac{\phi}{3} \right\rfloor \\ w_{\text{bifpn}} = w_{\text{head}} &= 64 \cdot 1.35^{\phi} \\ r &= 512 + 128 \cdot \phi \end{aligned} \tag{2.27}$$

Here, r is the resolution of the input image in both dimensions. The width parameters w_{bifpn} and w_{head} are the same for the FPN and the detector head. The depth parameters d_{bifpn} and d_{head} vary however.

The input image is first processed by the EfficientNet backbone. It produces feature maps of different resolutions due to downsampling. These feature maps form a pyramid with feature levels P_i. Each level P_i corresponds to feature maps with a resolution of $1/2^i$ per dimension of the original input image.

The last feature maps produced by the backbone at feature levels P_3 to P_7 are fed to a Bidirectional Feature Pyramid Network (BiFPN). Feature levels P_6

and P_7 do not occur in the original EfficientNet backbone but are obtained for EfficientDet by two additional downsampling layers. Tan, Pang, and Le introduce the BiFPN structure to efficiently aggregate information from different feature levels. It contains multiple top-down and bottom-up augmentation paths. The structure of the BiFPN is visualized in Figure 2.10. Each block inside this module performs the following operations:

1. Scale all inputs feature maps to the desired output resolution (if necessary). This is done using bilinear interpolation, followed by a depthwise separable convolution and a Batch Normalization layer.
2. Weight the resized input feature maps f_i according to the following formula:

$$\tilde{f}_i = f_i \cdot \frac{w_i}{\varepsilon + \sum_j w_j}$$

   Here, $w_i \in \mathbb{R}_{\geq 0}$ are trainable parameters and ε is a small constant to avoid division by zero.
3. Calculate the element-wise sum of the weighted feature maps.
4. Perform a depthwise separable convolution on the result.
5. Apply Batch Normalization.
6. Apply the swish activation function.

Step 2 and 3 allow the BiFPN to learn which features contain valuable information at a certain resolution. Uninformative features can be down-weighted before calculating the sum.

Each output feature map of the BiFPN is then passed to an object detection head. The object detection heads are based on RetinaNet detection heads. They contain a series of d_{head} depthwise separable convolution layers, each followed by a Batch Normalization layer and a swish activation function.

Figure 2.11 shows the Average Precision (AP) over the number of FLOPs on the COCO dataset [Lin+14] for some of the presented object detection architectures. The EfficientDet architecture is scalable and achieves the highest performance given a computational constraint.

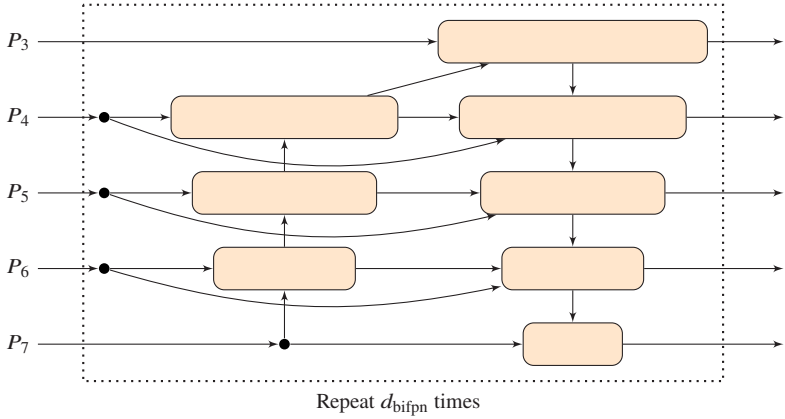


Figure 2.10: Conceptual structure and connectivity of the BiFPN proposed by Tan, Pang, and Le [TPL20]. The width of the blocks indicates the spatial resolution of the corresponding output feature maps. The displayed module is repeated d_{bifpn} times to form the complete BiFPN.

Alternative Approaches to Object Detection

Alternative object detection approaches have been published that are not based on anchor boxes. So far, however, they do not clearly outperform anchor-based approaches.

Some of these approaches describe objects as pairs of bounding box corners [LD18; Law+19], by a pair of corners and the center point [Dua+19], or by one point on each of the four sides of the bounding box [ZZK19]. The task then is to detect and group these keypoints. Other approaches classify center pixels of objects and then regress the bounding box size [ZWK19] or the distances to the four bounding box sides [Tia+19]. Yang et al. describe the outline of an object by a number of points, given as offsets from the center location [Yan+19]. In a previous work, we also performed instance segmentation by predicting object center pixels and pixel-level neighbor relations [SL18].

It is also possible to view the object detection task as a direct set prediction problem. It can then be solved using e. g. Transformers [Car+20] or RNNs [SAN16; RT16].

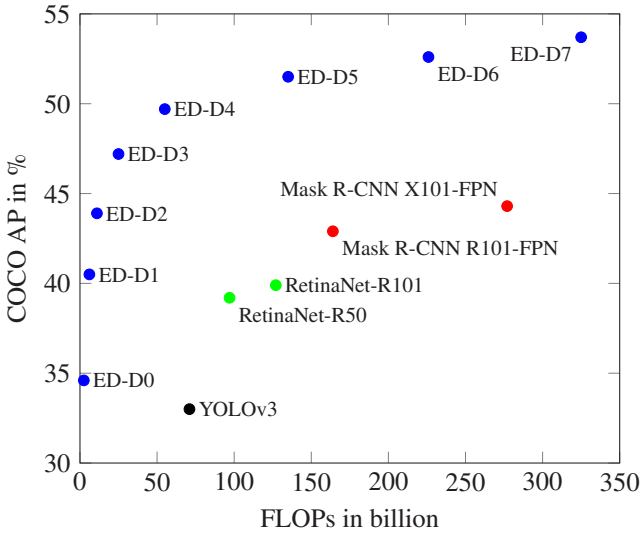


Figure 2.11: AP over FLOPs on the COCO "test-dev" ("val" for Mask R-CNN) dataset [Lin+14] for different object detection architectures [TPL20]. ED-D0 to ED-D7 is short for EfficientDet-D0 to EfficientDet-D7.

2.3.4 Optical Flow Architectures

According to Horn and Schunck, "optical flow is the distribution of apparent velocities of movement of brightness patterns in an image" [HS81]. In other words, the optical flow between two consecutive images of a video stream is a vector field that, for each pixel, describes the displacement of the surrounding image patch from one image to the other. Variational methods based on the work of Horn and Schunck have been popular for a long time to calculate optical flow. But optical flow can also be estimated using CNNs.

FlowNet [Dos+15b] is one of the first approaches that successfully estimate optical flow with a CNN. The authors propose two variants, FlowNetS and FlowNetC. FlowNetS stacks the two input images on top of each other and then processes them with a CNN backbone. The backbone successively reduces the resolution of the feature maps. After the backbone, they are successively

upsampled again in the decoder. Each resulting feature level has a skip connection from the backbone to the decoder in order to restore finer details. Optical flow is predicted at each resolution level in the decoder. If an optical flow estimate is already available from a coarser level, it is upsampled and used as input for the estimation at the next level. This way, each level can refine the estimate from the previous level.

The FlowNetS architecture must learn to perform the feature matching inside the convolutional layers. FlowNetC, on the other hand, uses a correlation layer to help with this task. In FlowNetC, the first three convolutional layers of the backbone are duplicated, so that the two input images can be processed separately. The resulting feature maps of both streams are then correlated by a correlation layer. The maximum displacement is limited when calculating the correlation to make the computation more tractable. The output of the correlation layer is a cost volume, which is then concatenated with the input feature maps. The resulting tensor is processed by the remaining part of the network, which is identical to FlowNetS. FlowNetC achieves a lower error than FlowNetS but is also computationally more expensive.

FlowNet2 [Ilg+17] stacks one FlowNetC and two FlowNetS networks on top of each other. The input images to the FlowNetS sub-networks are warped based on the flow estimation of the preceding sub-network. Therefore, the task of each FlowNetS sub-network is to correct the errors that were made by the previous stage. FlowNet2 also incorporates a fourth sub-network that is tuned to predict small displacements. The results of both branches are fused for the final result.

SPyNet [RB17] is a pure CNN architecture like FlowNetS. Its key contribution is to employ a spatial pyramid network: The backbone generates a feature pyramid for each input image separately. Then, optical flow is estimated at the highest feature level (with the lowest resolution) by a sub-network. The flow estimation is then upsampled to match the resolution of the next feature level. When resampling a flow image, also the magnitude has to be scaled by the same factor. The corresponding feature maps of the second image are now warped by the flow estimate. If the estimate is perfect, the result will align with the corresponding feature maps of the first image. In practice, the estimate will however not be perfect. Another sub-network therefore estimates the residual flow between the warped feature maps and the corresponding feature maps of the first image. The refined flow estimate is then obtained by adding the

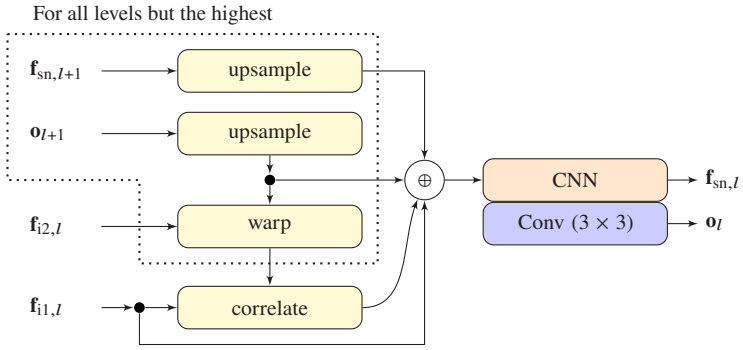


Figure 2.12: Structure of the flow estimation sub-network from PWC-Net [Sun+18]. At the highest level of the feature map pyramid, there is no flow estimate $\mathbf{o}_{l+1}$, and no previous sub-network features $\mathbf{f}_{\text{sn},l+1}$ with lower resolution. In this case, the upsampling and warping layers are omitted. Instead, the feature maps $\mathbf{f}_{\text{i1},l}$ and $\mathbf{f}_{\text{i2},l}$ of both images are directly fed into the correlation layer. The output of the flow estimation sub-network consists of the refined flow estimation $\mathbf{o}_l$ and the sub-network features $\mathbf{f}_{\text{sn},l}$. The CNN block in this figure contains convolution layers with DenseNet [Hua+17] connections.

residual flow estimate to the upsampled flow estimate. This process is repeated until a flow estimate at the original image resolution is obtained.

PWC-Net [Sun+18] combines the spatial pyramid and warping approach from SPyNet with the cost volume from FlowNetC. Also the backbone of PWC-Net generates a feature pyramid for each input image separately. Then, the same successive refinement steps as in SPyNet are executed: Starting from the highest feature level with the lowest resolution, a sub-network estimates the flow at this resolution. The flow is upsampled and then passed to the flow estimation sub-network at the next level. The input feature maps are warped by the upsampled flow estimate. The residual flow is calculated and used to refine the estimate. These steps continue until the flow estimate at the final resolution is obtained.

The sub-network of PWC-Net for flow estimation is however different from SPyNet and incorporates a correlation layer. It is visualized in Figure 2.12. Both the optical flow estimate $\mathbf{o}_{l+1}$ and the output feature maps $\mathbf{f}_{\text{sn},l+1}$ of the previous sub-network at feature level $l+1$ are upsampled to match the resolution of the current feature level l. Then, the corresponding backbone feature maps of

the second input image $\mathbf{f}_{\mathrm{i2},l}$ are warped by the upsampled flow. These steps are omitted if there is no higher feature level, i. e. if $\mathbf{o}_{l+1}$ and $\mathbf{f}_{\mathrm{sn},l+1}$ are not available. The warped feature maps (or $\mathbf{f}_{\mathrm{i2},l}$ if the step was omitted) are then correlated with the backbone feature maps of the first input image $\mathbf{f}_{\mathrm{i1},l}$. The output of this is a cost volume. Because of the pyramid structure and the successive refinement steps, the maximum displacement at each level cannot be large. The search distance can therefore be limited, which makes the calculation of the cost volume efficient. Finally, the cost volume, the upsampled optical flow, and some feature maps (c. f. Figure 2.12) are concatenated and processed by an CNN. The output of the module is the refined optical flow estimate $\mathbf{o}_l$ for feature level l, as well as sub-network feature maps $\mathbf{f}_{\mathrm{sn},l}$.

Sun et al. report that PWC-Net is faster than the previously presented methods while achieving a lower error.

All presented models were trained in a supervised manner. Ground truth data can easily be obtained for synthetic datasets like the FlyingChairs dataset, which was released to train FlowNet [Dos+15a]. Despite the dataset being synthetic, the trained networks often generalize well. Depending on the application, the performance can be improved by an additional fine-tuning step on domain-specific data.

It is also possible to learn optical flow in a self-supervised manner [YHD16]. This is typically done by feeding two consecutive images of a video stream to the flow estimation network. One input image is then warped by the estimated optical flow to reconstruct the other. The training objective is then to minimize the photometric error between the reconstructed and the real image. While this approach achieves good results, it does not outperform supervised learning.

A similar approach is used by Godard et al. to learn monocular depth estimation [God+19]. They use the ℓ^1 and SSIM [Wan+04] similarity metrics to calculate the photometric error and automatically mask unobservable pixels during training. Their approach achieves state of the art performance on the KITTI 2015 dataset [GLU12].

2.4 Tracking

The task of object detection is to find objects in a single image. The task of object tracking, on the other hand, is to estimate the position of an object over time. For this, tracks have to be maintained and incoming detections have to be assigned to tracks. In the case of multiple objects, the term Multiple Object Tracking (MOT) is commonly used.

Tracking approaches usually consist of an association (or re-identification) method and a motion model. The former is responsible for localizing previously observed objects in the current video frame. The latter is used to suppress detection noise, to ensure a physically plausible association, and to predict the state of existing objects. The prediction is especially important when no new measurements are available for a tracked object.

Based on the association approach, tracking algorithms can be classified as being detection-free or detection-based. Detection-free algorithms are initialized with a set of object locations. The algorithm then tries to find the same objects in all consecutive frames based on visual features. A well-known method for this is the correlation filter [Bol+10; Hen+15; Dan+15]. Similarly, optical flow can be used to track the position of an object for a limited amount of time. It is also possible to use siamese networks for this task [Ber+16; DS18; He+18; Li+18].

Detection-based methods became popular with the emergence of very accurate object detectors. Here, an object detector generates a list of detections for each video frame. These detections are then associated with existing tracks. This can e. g. be done based on position, bounding box IoU, or visual features. The advantage of detection-based algorithms is, that they solve the track initialization and association step simultaneously. Because of that, they tend to be fast and computationally efficient. Detection-free algorithms, on the other hand, would still need to run an object detector to initialize tracks. We, therefore, focus on detection-based methods in this work.

A tracking approach can either be an online or an offline approach. Offline approaches do not need to meet any real-time requirements. But more importantly, they have access to the whole video sequence. This makes acausal reasoning possible. Graph-based methods that optimize the whole sequence at once can achieve high accuracies, and are very popular [ZLN08; PRF11; BC13].

Given the intended application, we are however only interested in online tracking approaches. They must use little computational resources and only perform causal computations. They can only access observations up to the present time and have to gradually extent existing trajectories.

Advanced tracking approaches can also include methods to deal with occlusions. Some approaches divide the object of interest into parts and only track the visible parts. It is also possible to store multiple visual appearance models per track. This allows to maintain one model for the occluded object, and one for the unoccluded object. An occluded object can then be more easily re-identified once it is not occluded anymore.

Full occlusions for a long time are challenging for tracking: Simple motion models like the constant velocity model with a Kalman filter [Kal60] work well for short prediction horizons. But they fail for long prediction horizons of multiple seconds. The velocities of the objects can change significantly during that time. An alternative is to use more sophisticated prediction methods to handle long occlusions, which lead to long prediction horizons. These prediction methods might output multiple hypotheses, and can take the interaction of agents as well as the physical feasibility into account.

In this work, we refrain from implementing these more advanced methods. They are more closely related to prediction than to perception. We want to highlight the strength of an accurate object detector, which can also handle short-time occlusions when used in a tracking-by-detection approach. Our presented ideas are however orthogonal to these methods, and they can be easily integrated into our approach if desired.

2.4.1 Tracking-by-Detection Approaches

Bewley et al. show in an early work [Bew+16] that a simple tracking approach with a strong object detector can achieve state of the art performance. They use a constant velocity motion model with a Kalman filter [Kal60]. The data association is done only based on the IoU between the predicted bounding boxes from the motion model and the current detections.

Based on this, Wojke, Bewley, and Paulus propose an extension that integrates appearance information into the association step [WBP17]. This reduces the

number of identity switches by 45 %. Also, instead of using the IoU between the predicted bounding boxes and detections, they use the Mahalanobis distance between the predicted filter states and the detections.

Feichtenhofer, Pinz, and Zisserman use a two-stage object detector to predict object bounding boxes as well as object displacements [FPZ17]. As usual for two-stage detectors, each frame is processed individually by the RPN, and then by the detection network in the second stage. The authors then introduce an additional second stage for displacement estimation. It calculates a correlation map between the feature maps of the current and the previous frame. During the RoI pooling step, a crop of the correlation map and the feature maps of both frames is generated for each proposal. Based on this information, the additional second stage estimates the displacement of each object between both frames. Tracks are then generated based on the IoU between the predicted bounding boxes from the old frame, and the current detections.

Luo, Yang, and Urtasun propose a tracking approach for lidar sensors that is based on a Bird Eye View (BEV) grid map [LYU18]. First, they perform ego-motion compensation of the last n BEV frames. This is important to keep the displacements of objects in the grid map small. Then they stack the grid maps and process them with a CNN. The CNN then produces a list of object detections as well as a predicted future trajectory per object. Objects are associated with existing tracks based on the overlap of the predicted trajectories.

Bergmann, Meinhardt, and Leal-Taixé propose Tracktor [BML19]. It uses the regression head of Faster R-CNN to regress a new bounding box for each given detection from the previous frame. If the position of the object has changed only slightly, the regression head is able to regress to the new position. After this step, new detections are associated with the existing tracks based on the bounding box IoU. The authors also propose two extensions to deal with larger object displacements: The first proposal is to use a motion model, and the second is to incorporate appearance information into the association step.

2.4.2 Motion Models

A widely used motion model is the Kalman filter [Kal60] with either a constant velocity or constant acceleration assumption. The performed computations are simple and fast, which makes it suitable for online approaches. It is an optimal

filter if the linear model matches the reality, and if the occurring errors are uncorrelated and Gaussian distributed with zero mean.

It is obvious that a constant velocity or constant acceleration assumption does not match reality. But in practice, it is a good approximation for short prediction horizons. Since the maximal acceleration of road users is physically limited, their velocities cannot change too much between two consecutive frames of a video (assuming reasonable frame rates).

It is also possible to use neural networks to learn a motion model. Most existing work is based on RNNs, and more specifically based on Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells [Gho+17; MBR17]. But it is also possible to combine neural networks with a Kalman filter: Coskun et al. propose to learn the nonlinear transition function of an Extended Kalman Filter (EKF) as well as the two noise covariance matrices using LSTM cells [Cos+17].

2.5 Evaluation Metrics

This section presents the evaluation metrics that we use in Chapter 3 and Chapter 4 to evaluate our approach. These metrics are commonly used in literature and allow to easily compare our approach to others.

2.5.1 Pixel-wise Semantic Segmentation

The semantic segmentation task is evaluated using the Jaccard Index, which is commonly referred to as PASCAL VOC IoU [Eve+15]. For each class c, it is calculated as follows:

$$\mathrm{IoU}_c = \frac{n_{\mathrm{TP},c}}{n_{\mathrm{TP},c} + n_{\mathrm{FP},c} + n_{\mathrm{FN},c}} \tag{2.28}$$

The mIoU is then the mean over the IoUs of all classes. Here, $n_{\mathrm{TP},c}$ is the number of true positive pixels, $n_{\mathrm{FP},c}$ is the number of false positive pixels and $n_{\mathrm{FN},c}$ is the number of false negative pixels. When evaluating the IoU for a class c, and observing a pixel with predicted class $\tilde{c}$ and ground truth class c^*, the following three cases can occur:

- $c = \tilde{c} = c^*$: Pixel counts as true positive.
- $c = \tilde{c} \neq c^*$: Pixel counts as false positive.
- $c = c^* \neq \tilde{c}$: Pixel counts as false negative.

2.5.2 Object Detection

For the object detection task, we report the AP at an IoU threshold of 0.5. This means that a detection only counts as a true positive if the IoU between its bounding box and a ground truth annotation is at least 0.5. We also report the Log-Average Miss Rate (LAMR) [Dol+17].

In order to calculate these metrics, first the number of true positives n_{TP}, false positives n_{FP}, and false negatives n_{FN} has to be determined. For this, detection bounding boxes have to be assigned to ground truth bounding boxes. This assignment is performed based on the IoU between the bounding boxes. A detection can be assigned to a ground truth bounding box if the IoU is larger than the desired threshold. If multiple detections can be assigned based on this criterion, then only the detection with the highest IoU is assigned. The IoU between two bounding boxes is calculated as follows:

$$\text{IoU}(\mathbf{b}_1, \mathbf{b}_2) = \frac{\text{area}(\mathbf{b}_1 \cap \mathbf{b}_2)}{\text{area}(\mathbf{b}_1 \cup \mathbf{b}_2)} \tag{2.29}$$

As the name suggests, the IoU is the area of the intersection of both bounding boxes divided by the area of the union of both bounding boxes.

With this, precision and recall can be calculated as follows:

$$\begin{aligned} \text{precision} &= \frac{n_{\text{TP}}}{n_{\text{TP}} + n_{\text{FP}}} \\ \text{recall} &= \frac{n_{\text{TP}}}{n_{\text{TP}} + n_{\text{FN}}} \end{aligned} \tag{2.30}$$

The set of bounding box detections changes for different confidence score thresholds. A high threshold means that only detections with high confidence scores are generated, while a low threshold results in more detections with lower confidence scores. Because of this, also precision and recall depend on

the confidence score threshold. A precision-recall curve can then be created by plotting the precision over the recall for different thresholds.

The area under this precision-recall curve gives the AP. It is therefore also referred to as Area Under Curve (AUC). PASCAL VOC uses this metric to evaluate bounding box detectors [Eve+15]. Please note that some publications use an interpolated precision-recall curve. This can result in slightly different AP values. The AP is a single number that allows to easily compare different precision-recall curves. If there are multiple classes of object, the AP can be calculated for each class. The Mean Average Precision (mAP) is then the mean over all classes.

Another possibility to compare detectors is to create a log-log plot of the miss rate over the number of False Positives Per Image (FPPI). The LAMR metric is then computed by by averaging miss rates at 9 FPPI values evenly spaced in log-space between 10^{-2} and 10^{0} [Dol+17]. The IoU threshold used for this metric is 0.5.

2.5.3 Tracking

We employ the widely used Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP) metrics [BS08] to evaluate our tracking approach. The first step to calculate these metrics is to match the object hypotheses of the tracker to the ground truth objects. The MOTP metric is calculated as the average localization error for all correctly matched object-hypothesis pairs. It indicates how well the tracker can estimate precise object locations, independent of its ability to associate detections to tracks.

The MOTA metric, on the other hand, indicates how well objects are detected and associated to tracks, independent of the exact localization performance. It is calculated as:

$$\text{MOTA} = 1 - (r_{\text{m}} + r_{\text{fp}} + r_{\text{mme}}) \tag{2.31}$$

Here, r_{m} is the miss ratio. It is the number of missed objects divided by the total number of ground truth objects. The term r_{fp} is the false positive ratio. It is the number of false detections divided by the total number of ground truth objects. Finally, r_{mme} is the mismatch ratio. It is the number of detections matched to the wrong track divided by the total number of ground truth objects.

3 Single-Frame Environment Perception

The goal of this work is to present a fast, efficient, and precise CNN architecture for video-based environment perception for self-driving cars. This chapter presents a CNN architecture that works on single camera frames. In Section 3.1, we will describe an architecture that can detect objects and perform pixel-wise semantic segmentation at the same time. This is computationally less expensive than evaluating two separate neural networks. In Section 3.2, we will present an extension to this architecture that uses a similarity metric to perform NMS.

Training and evaluation are performed on the Cityscapes dataset [Cor+16] and the BDD100K dataset [Yu+20]. The Cityscapes dataset contains semantic segmentation and instance segmentation masks. We use the former directly to train the segmentation task and convert the latter to bounding box annotations to train the object detection task. The BDD100K dataset consists of multiple sub-datasets. We use the segmentation dataset to train the segmentation task, and the tracking dataset to train the object detection task. We mix both of these to train our MTL architecture.

Both the Cityscapes dataset and the BDD100K dataset are split into a training, validation, and testing set. The ground truth annotations for the testing sets are however not publicly available. We therefore cannot run an evaluation script on the test sets but have to rely on the official test servers. The metrics that are reported by the test servers are however fixed and not consistent between datasets. In order to report consistent metrics across different datasets, we evaluate our approach on the validation sets. This is valid because we did not perform any hyperparameter tuning on the official validation sets. Instead, we used a set of images that we kept out from the training sets for hyperparameter tuning.

We use the FlyingChairs dataset, which was released to train FlowNet [Dos+15a], to train our sub-network for optical flow estimation in Chapter 4.1. We pretrain our backbone on the WebVision 1.0 dataset [Li+17].

We use TensorFlow [Aba+16] to implement our models. Run-time measurements are carried out on an *Nvidia Quadro RTX 8000* GPU in mixed-precision mode. We choose this GPU because it is expected that Nvidia's upcoming *DRIVE AGX Orin* platform for automotive applications will achieve comparable performance [Lab19]. This shows that series cars will soon have access to enough computational resources to run our proposed approach online.

Unless noted otherwise, we use the following hyperparameters to train our models:

- AdamW optimizer (Adam with decoupled weight decay) [KB15; LH19]
- Batch size of 8
- Initial weight decay of 10^{-7}, scaled proportional with learning rate
- Initial learning rate of 10^{-3}, two decays by a factor of 10 each
  - First learning rate decay after 600 000 steps
  - Second learning rate decay after 1 100 000 steps
- 1 800 000 training steps in total
- Frozen Batch Normalization statistics during the last 300 000 steps
- Training on image crops with resolution of 512×256

We use Batch Normalization [IS15] as normalization approach throughout our models. But we found that Online Normalization [Chi+19] can also achieve good results if the training hardware does not allow for a large enough batch size.

3.1 Simultaneous Object Detection and Semantic Segmentation

This section presents a CNN architecture for simultaneous object detection and semantic segmentation. Our evaluation shows how the tasks influence each other in the MTL setting.

Our proposed network does not perform instance segmentation. We argue that a bounding box object representation is good enough for most downstream consumers in the software stack of a self-driving car. But if instance segmentation masks are desired, it is possible to use the Mask R-CNN instance segmentation head together with our approach.

It is obvious that self-driving cars must be able to detect objects. The behavior generation and planning modules must know about the existence and location of other road users to reason about them. Otherwise, it is not possible to plan a collision-free path. Also, the detection of traffic signs and traffic lights is important so that the displayed information can be taken into account in the planning stages.

But pixel-wise semantic segmentation is important for self-driving cars, too. It can, for example, be used to extract all pixels belonging to the road surface. This information can be used to validate that the planned trajectory lies in the drivable area. It can also happen that a self-driving car does not have access to up-to-date maps of the current area, e. g. inside a frequently changing construction site. In these cases, lanes have to be detected online. This task can be solved using pixel-wise semantic segmentation, too [Mey+18]. Visual localization relies on storing the positions of landmarks in a map. These landmarks are points that can be easily detected in camera images. They can be used for localization when detecting them again during successive drives inside the mapped area. It makes sense to only store and detect landmarks that are part of the static environment. Semantic segmentation can be used to extract the static parts of the scene in the camera image.

3.1.1 Network Structure

When designing an MTL architecture for simultaneous object detection and semantic segmentation, it is possible to start with an architecture for semantic segmentation and modify it to also perform object detection, or the other way around. As shown in Section 2.3.2 and Section 2.3.3, there is a high variability in successful semantic segmentation architectures. Object detection architectures, on the other hand, have converged to a few common design patterns. Therefore, it makes sense to base the design on an object detection architecture.

This work focuses on fast approaches that can meet the real-time requirements of self-driving cars. Because of this, we want to use a single-stage detector. The semantic segmentation task relies on fusing information from different levels of detail to achieve good accuracy. Therefore an object detection architecture with an FPN is an obvious choice as a starting point.

We base our MTL architecture on the EfficientDet [TPL20] architecture. It is to date the best-performing object detection architecture that fulfills all requirements. Also, the authors mention that a slightly modified version of EfficientDet can achieve reasonable semantic segmentation results. This choice is further supported by a recent publication [Kir+19]. It presents an architecture for simultaneous instance segmentation and semantic segmentation that also employs an FPN. The overall structure is similar to the one proposed in this section.

We use the EfficientDet-D3 backbone in our experiments. The D3 version represents a good compromise between accuracy and computational demand. All results reported in the following sections will be based on this version. But the whole architecture is scalable, and a different compound scaling factor can be selected based on the requirements.

According to equation (2.27), the image resolution for D3 should be selected as $r = 512 + 128 \cdot 3 = 896$ pixels per dimension. This results in a total of $896^2 = 802\,816$ pixels in the input image. But typical camera images as used in a self-driving car do not have an aspect ratio of 1:1. The images of the Cityscapes dataset, for example, have an aspect ratio of 2:1, while the ones of the BDD100K dataset have an aspect ratio of 16:9. We therefore use a

resolution of $1\,280 \times 640$ pixels during inference. This keeps the total amount of pixels, and thus also the computation time, roughly the same.

We change the BiFPN to use feature levels P_2 to P_7 (c. f. Section 2.3.3). The inclusion of P_2 unfortunately increases the computational demands of the BiFPN significantly. But our experiments showed that this change is important to recover fine image details for the pixel-wise semantic segmentation. The inclusion of P_2 can also help for object detection in the context of self-driving cars: Some relevant objects like distant traffic lights are too small in the input image to be detected on level P_3 or higher.

Our proposed semantic segmentation network head is as simple as possible. It has the same structure as the standard network heads in EfficientDet-D3. That is, it consists of a series of depthwise separable convolutions, each followed by a Batch Normalization layer, and a swish activation function. The general structure of these network heads is visualized in Figure 3.1. In the case of the network head for semantic segmentation, the last layer is a transposed convolution. It upsamples the feature maps by a factor of 4 to match the resolution of the input image. At the same time, it reduces the number of channels to the number of semantic classes. We attach the semantic segmentation head only to the P_2 level of the BiFPN since we are not interested in predictions with lower resolutions.

We use three different network heads for object detection. Again, each has the same structure as the standard network heads of EfficientDet-D3. The last layer of each is a 1×1 convolution layer that adjusts the number of channels to the required number of output channels.

This is different from RetinaNet and the original EfficientDet which use only two network heads. Both our and their designs have one box regression head that predicts four parameters per anchor box. These are calculated according to equation (2.25). The second network head of RetinaNet predicts a score per object class for each anchor box. In other words, it independently estimates probabilities $\Pr(\text{object}, \text{class} = c_i)$ for all classes c_i and for each anchor box.

We argue that this is not a good choice in the context of self-driving cars. Using the definition of conditional probability, the equation above can be rewritten as:

$$\Pr(\text{object}, \text{class} = c_i) = \Pr(\text{object}) \cdot \Pr(\text{class} = c_i | \text{object}) \tag{3.1}$$

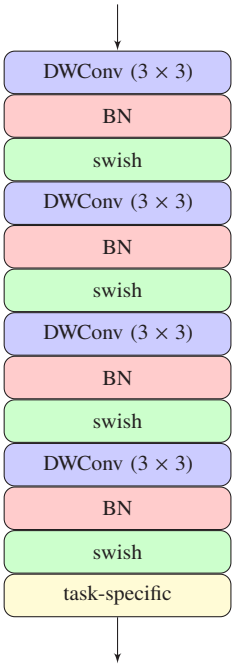


Figure 3.1: The network head that is attached per task at each feature level. The last layer is task-specific. It ensures the right resolution and number of channels in the output.

Let's assume that we are interested in an object detector for two-wheelers, and that there are two classes—motorcycles and bicycles. Let there be an image with a two-wheeler in it, and let's assume that this two-wheeler is detectable as such. The existence probability might be $\Pr(\text{object}) = 0.9$ in this example. Now assume that the image quality is not good enough to decide the exact object class. It might be a mountain bike with thick tires and frame or an entry-level motorcycle. The estimated class probabilites might be

$\Pr(\text{class} = \text{bicycle}|\text{object}) = 0.5$ and $\Pr(\text{class} = \text{motorcycle}|\text{object}) = 0.5$. For this example, it follows that:

$$\begin{aligned}\Pr(\text{object}, \text{class} = \text{bicycle}) &= \Pr(\text{object}) \cdot \Pr(\text{class} = \text{bicycle}|\text{object}) \\ &= 0.9 \cdot 0.5 = 0.45 \\ \Pr(\text{object}, \text{class} = \text{motorcycle}) &= \Pr(\text{object}) \cdot \Pr(\text{class} = \text{motorcycle}|\text{object}) \\ &= 0.9 \cdot 0.5 = 0.45\end{aligned}$$

The predicted scores are compared against a threshold, and all detections with lower existence probability are discarded. If we use the common threshold of 0.5 in this example, then the detector would not detect any object. This, however, would be fatal.

We therefore propose to predict both factors of equation (3.1) separately. Accordingly, we add two network heads that predict one factor each. In the previous example, our detector would detect the object, but it would be unsure about the classification. A wrong classification is not good since it can e. g. cause inaccuracies in the behavior prediction of a self-driving car. But it is definitely better than not detecting the object.

We also extend the set of anchor boxes. The RetinaNet detector uses anchor boxes of aspect ratios $\{1{:}2, 1{:}1, 2{:}1\}$. We add two more aspect ratios and use $\{1{:}4, 1{:}2, 1{:}1, 2{:}1, 4{:}1\}$. The reason is that we also want to detect more asymmetric objects like trams or pedestrians.

We train all classification network heads using Focal Loss [Lin+17c] with $\gamma = 1.5$ to account for the class imbalances. We train the bounding box regression network head using smooth ℓ^1 loss with $\delta = 0.1$. We use the weighting approach proposed by Kendall, Gal, and Cipolla [KGC18] to weight the different task losses. The overall network structure is visualized in Figure 3.2.

3.1.2 Data Augmentation

In general, CNNs profit from a lot of training data. More training data means better generalization and higher accuracy. One reason for this is that it reduces overfitting. Overfitting means that the model just remembers previously seen examples, but does not generalize.

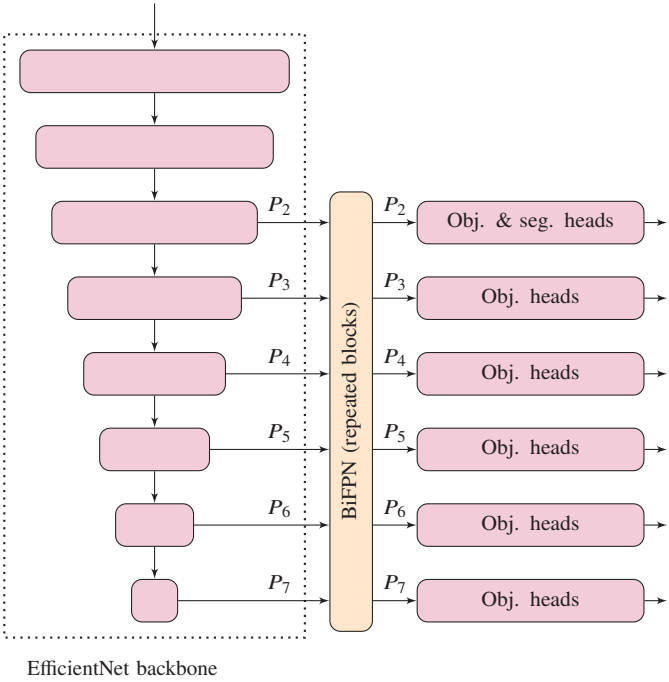


Figure 3.2: Overall structure of our proposed network for simultaneous object detection and semantic segmentation. Each block performs complex operations: The EfficientNet backbone on the left side consists of MBConv blocks as visualized in Figure 2.6. They are arranged as depicted in Table 2.1, but scaled by equation (2.24). The BiFPN consists of multiple repeated blocks according to equation (2.27). Their structure is visualized in Figure 2.10. The structure of each network head is visualized in Figure 3.1.

Annotating training data is however a cumbersome and expensive process. Even though larger and larger datasets for automated driving became available during the last few years, the amount of labeled training data is limited. One alternative to annotating more data is to augment the annotated training data in a way that does not invalidate the annotations. This increases the number of available training examples artificially. The relevant information contained in the images must not be altered by the augmentation, but the visual appearance of the image can change. An alternative view is that augmentation adds noise to the training images, which can have a regularizing effect and improve generalization performance.

We apply the following distortions to the training images as part of our augmentation strategy. For this, we represent the pixel intensities of the image as floating point values between 0 and 1:

1. Random crop of the image. The crop is performed in a way so that the size of objects (measured in pixels) during training matches the size of objects during inference. This is ensured even if the resolution and the Field of View (FoV) of the training images is different from the images used for inference. Details will be provided at the end of this section.

2. Random horizontal flip of the image. This operation must only be performed if it does not change the semantics of the image. In some cases, this does not hold, e. g. when classifying lanes as being a left or a right turn lane. The flip is performed with a 50 % chance:

$$\mathbf{I}_{\mathrm{RGB},XY} \leftarrow \begin{cases} \mathbf{I}_{\mathrm{RGB},XY} & \text{if } d < 0.5 \\ \mathbf{I}_{\mathrm{RGB},-XY} & \text{otherwise} \end{cases}, \quad d \sim \mathcal{U}(0,1) \tag{3.2}$$

3. Random change of the gamma curve. The exponentiation is performed element-wise, with the same γ value for each pixel of the image:

$$\mathbf{I}_{\mathrm{RGB},XY} \leftarrow \mathbf{I}_{\mathrm{RGB},XY}^{\gamma}, \quad \gamma \sim \mathcal{U}(0.8, 1.2) \tag{3.3}$$

4. Random change of contrast:

$$\mathbf{I}_{\mathrm{RGB}} \leftarrow c \cdot \mathbf{I}_{\mathrm{RGB}}, \quad c \sim \mathcal{U}(0.8, 1.2) \tag{3.4}$$

5. Random change of brightness:

$$\mathbf{I}_{\mathrm{RGB}} \leftarrow \mathbf{I}_{\mathrm{RGB}} + b, \quad b \sim \mathcal{U}(-0.2, 0.1) \tag{3.5}$$

6. Random change of hue:

$$\begin{aligned} \mathbf{I}_{\mathrm{HSV},XY} \leftarrow \mathbf{I}_{\mathrm{HSV},XY} + (h, s, 0)^T, \quad & h \sim \mathcal{U}(-10, 10), \\ & s \sim \mathcal{U}(-0.1, 0.1) \end{aligned} \tag{3.6}$$

   For this, the image is first converted from the RGB color space to the HSV color space, and afterwards converted back.

7. Random change of white balance:

$$\begin{aligned} \mathbf{I}_{\mathrm{Lab},XY} \leftarrow \mathbf{I}_{\mathrm{Lab},XY} + (0, a, b)^T, \quad & a \sim \mathcal{U}(-5, 5), \\ & b \sim \mathcal{U}(-5, 5) \end{aligned} \tag{3.7}$$

   For this, the image is first converted from the RGB color space to the CIELAB color space, and afterwards converted back.

8. Gaussian blur of the image by performing a convolution with the symmetric Gaussian function G_σ:

$$\mathbf{I}_{\mathrm{RGB}} \leftarrow \mathbf{I}_{\mathrm{RGB}} * G_\sigma, \quad \sigma \sim \mathcal{U}(0, 1.3) \tag{3.8}$$

9. Add noise to each pixel in the image:

$$\begin{aligned} \mathbf{I}_{\mathrm{RGB},XY} \leftarrow \mathbf{I}_{\mathrm{RGB},XY} + n_{XY}, \quad & n_{XY} \sim \mathcal{N}(0, \sigma^2), \\ & \sigma \sim \mathcal{U}(0, 0.03) \end{aligned} \tag{3.9}$$

The parameters and the order of these pre-processing steps were determined empirically. The goal of this step is to create images with a high variance that cover all distortions which might be encountered during inference. This includes, for example, different sensor characteristics and a different white balance, higher sensor noise, or not perfectly focused optics. On the other hand, the generated images should still look mostly natural, and the important image information must not get lost.

The parameters from above result in an augmentation that is already a bit too strong to achieve the best results on the training dataset. But they improve results across different datasets with different cameras. This is important for the application in the research vehicle of MRT at KIT. It uses cameras that are different from the ones used to record the Cityscapes and BDD100K datasets. If best results on the training dataset are desired, the variances of the random distortions can be decreased a bit.

Our model is trained on random crops of the annotated training images (c. f. point one above). Objects in these crops must approximately have the same size (in pixels) as in the images used during inference. The reason is that CNN architectures are translation equivariant or even invariant by design, but not scale invariant. We perform a number of calculations to extract training crops that fullfil this requirement. By following these steps, we can train a CNN on a dataset with a narrow FoV and still achieve reasonable performance during inference on images with a much wider FoV.

We assume that all images are undistorted to match a spherical camera model where the poles of the sphere are above and below the car. Such an undistort is suitable for automated vehicles because it allows for a very wide horizontal FoV. At the same time, it does not violate the translation equivariance assumption of the CNN design—a small patch centered around a fixed point on an object does not change if the object moves in the image. A pinhole undistort would obviously violate this assumption since objects at the edges of the image are stretched. For smaller FoV, the assumption however holds approximately. In this case, a pinhole undistort can also be used. Equation (3.10) and equation (3.11) in the following procedure can then easily be modified by using trigonometric functions.

1. We calculate the appropriate horizontal FoV of the training image $\mathrm{FOV}_{\mathrm{training}}$ as follows:

$$\mathrm{FOV}_{\mathrm{training}} = \mathrm{FOV}_{\mathrm{inference}} \cdot \frac{w_{\mathrm{training}}}{w_{\mathrm{inference}}} \tag{3.10}$$

   This equation depends on the desired horizontal FoV during inference $\mathrm{FOV}_{\mathrm{inference}}$ and horizontal resolution $w_{\mathrm{inference}}$, as well as the horizontal resolution during training w_{training}. The parameters $w_{\mathrm{inference}}$ and $\mathrm{FOV}_{\mathrm{inference}}$ are given by the desired application. The parameter w_{training}

has to be chosen in a way that the final crop size is feasible and contains a large enough part of the image.

The FoVs and image widths in the formula above scale linear with each other. This becomes obvious when considering that the length s of a circular arc with radius r and angle θ is $s = \theta \cdot r$: The arc length scales linearly with the angle.

2. We now calculate the appropriate width of the crop $\bar{w}_{\text{crop}}$ following the same reasoning:

$$\bar{w}_{\text{crop}} = w_{\text{annotation}} \cdot \frac{\text{FOV}_{\text{crop}}}{\text{FOV}_{\text{annotation}}} \tag{3.11}$$

Here, $w_{\text{annotation}}$ is the width of the annotated images from the training dataset, and $\text{FOV}_{\text{annotation}}$ is their FoV. It also holds that $\text{FOV}_{\text{crop}} = \text{FOV}_{\text{training}}$ because the crop will be used for training.

3. We vary the width of the crop slightly as a data augmentation technique. We therefore sample the final width between 95 % and 105 % of the appropriate width that was calculated in the step before:

$$w_{\text{crop}} \sim \mathcal{U}\left(0.95 \cdot \bar{w}_{\text{crop}}, 1.05 \cdot \bar{w}_{\text{crop}}\right) \tag{3.12}$$

4. We calculate the corresponding height of the crop. This will preserve the aspect ratio in step 6:

$$h_{\text{crop}} = w_{\text{crop}} \cdot \frac{h_{\text{training}}}{w_{\text{training}}} \tag{3.13}$$

Again, the parameter h_{training} has to be chosen in a way that the final crop size is feasible and contains a large enough part of the image.

5. We sample the top-left corner of the crop uniformly so that all feasible locations are covered:

$$x \sim \mathcal{U}\left(0, w_{\text{annotation}} - w_{\text{crop}}\right), \quad y \sim \mathcal{U}\left(0, h_{\text{annotation}} - h_{\text{crop}}\right) \tag{3.14}$$

6. We perform the crop and resize the result to match the training width and height of w_{training} and h_{training}.

These steps are, of course, not only performed on the input images, but also on the corresponding training annotations. It means that the coordinates of bounding boxes, polygons describing the outlines of objects, and so on, are adjusted accordingly.

3.1.3 Target Assignment

Training datasets for object detection usually contain a list of bounding box annotations per image. In the case of the Cityscapes dataset, we generate this list from the polygon annotations of the object instances.

The direct output of our neural network, however, is not a list of bounding boxes but a set of predictions relative to the anchor boxes. Each discrete spatial location at the feature levels P_2 to P_7 has predictions for several anchor boxes of different sizes and aspect ratios. Each prediction consists of:

- An objectness score. It indicates how likely the anchor box corresponds to an object in the image.
- Class scores for each possible class of the object.
- Relative regression of box parameters describing the difference between the anchor box parameters and the real bounding box parameters (c. f. equation (2.25)).

During training, the prediction targets for these outputs have to be generated from the list of bounding box annotations. We generate these prediction targets as follows: First, we enumerate all anchor boxes corresponding to all output positions of the neural network. Each of these anchor boxes can either be in an `assigned` state, a `not assigned` state, or a `don't care` state. The `assigned` state indicates that the anchor box has a corresponding ground truth annotation. The `not assigned` state indicates that there is no corresponding ground truth annotation. The `don't care` state indicates that we do not want to enforce a certain output for this anchor box. We will mask all losses corresponding to these anchor boxes during training.

The `assigned` and `not assigned` states directly provide the ground truth for the objectness prediction target. The losses of all other prediction targets are

masked unless the corresponding anchor box is in the `assigned` state. In that case, the desired prediction targets are provided by the assigned ground truth bounding boxes.

After enumerating the anchor boxes, the IoUs between all anchor boxes and all ground truth annotations are calculated. Based on the result, anchor boxes are then assigned to bounding box annotations:

- If the highest IoU for an anchor box is larger than 0.5, the state of the anchor box becomes `assigned`.
- It the highest IoU for an anchor box is between 0.4 and 0.5, its state becomes `don't care`.
- If the highest IoU for an anchor box is below 0.4, its state becomes `not assigned`.

The second rule effectively masks all losses for IoU values between 0.4 and 0.5. This is done to make training more stable. Examples right at the decision threshold can otherwise introduce noise in the training process, and result in high loss values.

The rules above are simple, but some additional corner cases also have to be taken into account:

- Only one ground truth bounding box can be assigned to an anchor box. If multiple ground truth bounding boxes have an IoU larger than 0.5 with the same anchor box, then the ground truth box with the highest IoU is assigned.
  - If, however, the difference between the IoU values of the two ground truth annotations with highest IoU values is less than 0.2, we set the state to `not assigned`. We do this to avoid an oscillation of the box regression targets during training: If an anchor box has a high overlap with two objects, then it is not obvious for which the regression parameters should be calculated.
- If a ground truth bounding box is not assigned to any anchor box based on the rules above, it gets assigned to the anchor box with the highest IoU, as long as it is at least 0.4.

Figure 3.3: Color codes used in Figure 3.4 to Figure 3.9.

3.1.4 Evaluation

Before we can evaluate our proposed approach for simultaneous object detection and semantic segmentation, we have to establish a baseline. All modifications that we propose in this thesis will then be compared against this baseline. For this, we train an object detection and a semantic segmentation model separately. Each model has the same structure as our MTL model, but only the task-specific network heads are attached.

A comparison with other state of the art models based on the precision metrics is definitely valid, but not conclusive. Tan, Pang, and Le already showed that the EfficientDet architecture achieves outstanding performance for the object detection task [TPL20]. Our goal, however, is not to achieve the best possible accuracy at any cost, but to achieve the best accuracy given the computational constraints. We also want our model to generalize across camera setups. For best accuracy without these constraints, the compound scaling factor and therefore also the image resolution can be increased, and the augmentation parameters can be adjusted. A comparison of our MTL models to our baseline models is meaningful since we keep the compound scaling factor and the other hyperparameters fixed. Differences in the performance metrics are therefore only a result of the architectural changes.

We train and test our baseline models on both the Cityscapes and the BDD100K datasets. We also test the semantic segmentation model on the Cityscapes dataset after training it on the BDD100K dataset. We do this to evaluate how well the model generalizes across datasets. We also provide qualitative results

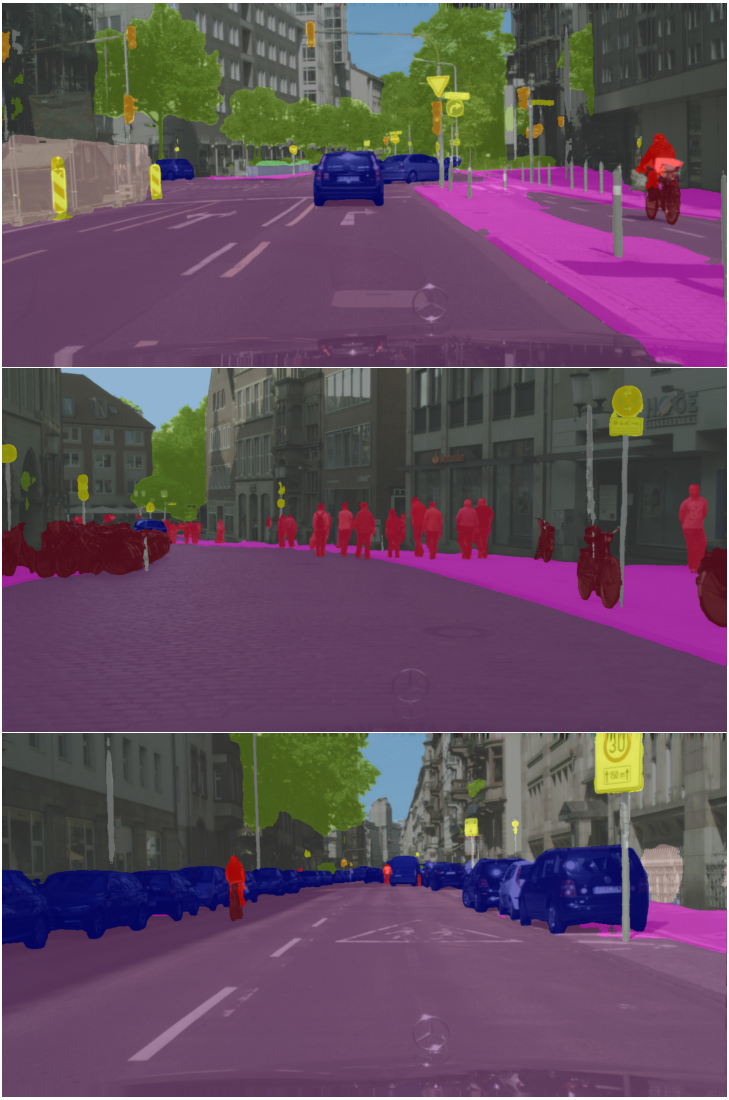


Figure 3.4: Qualitative results of the semantic segmentation task on the Cityscapes validation dataset. The CNN was trained on the Cityscapes training dataset. The corresponding color codes can be found in Figure 3.3.

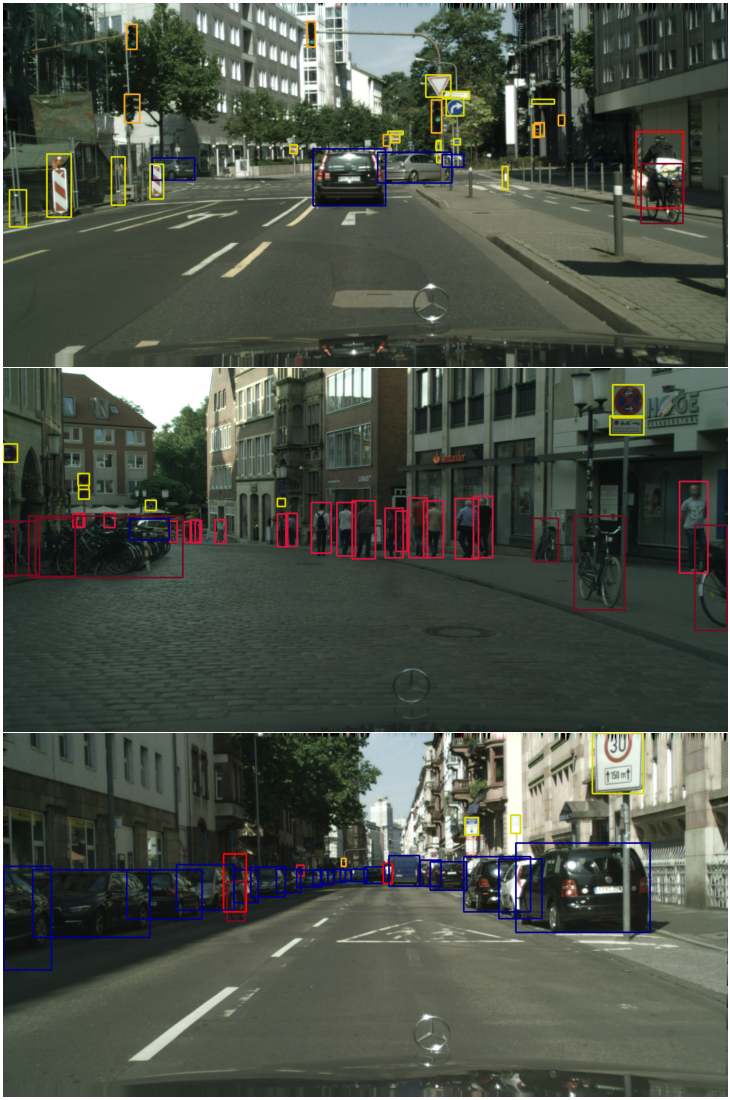


Figure 3.5: Qualitative results of the object detection task on the Cityscapes validation dataset. The CNN was trained on the Cityscapes training dataset. The corresponding color codes can be found in Figure 3.3.

Figure 3.6: Qualitative results of the semantic segmentation task on the BDD100K validation dataset. The CNN was trained on the BDD100K training dataset. The corresponding color codes can be found in Figure 3.3.

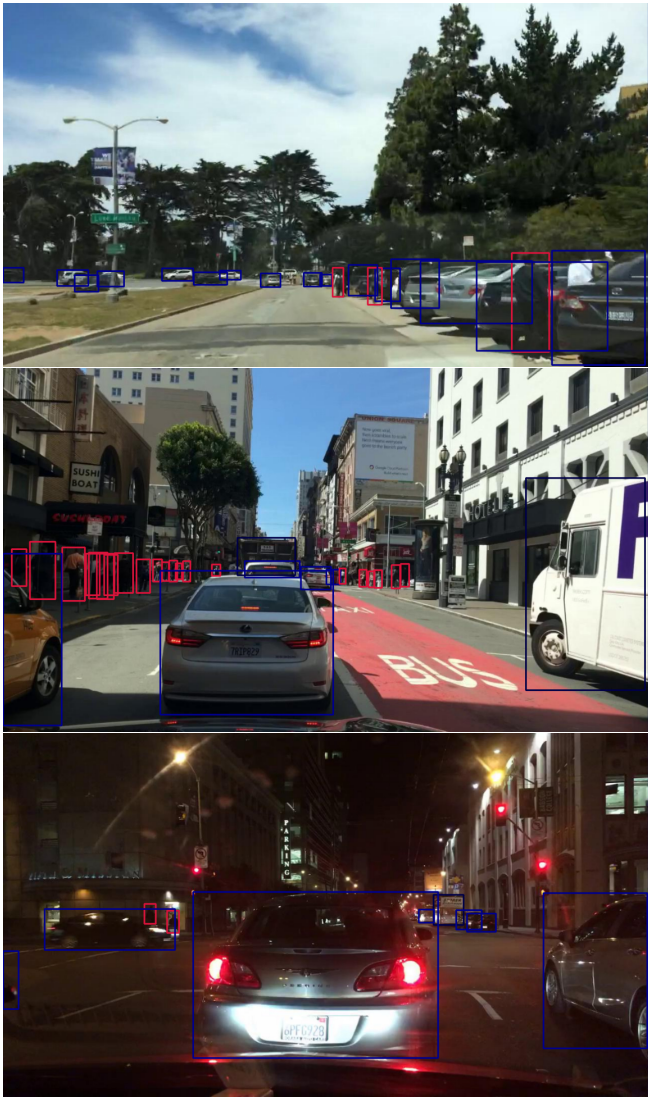


Figure 3.7: Qualitative results of the object detection task on the BDD100K validation dataset. The CNN was trained on the BDD100K training dataset. The corresponding color codes can be found in Figure 3.3.

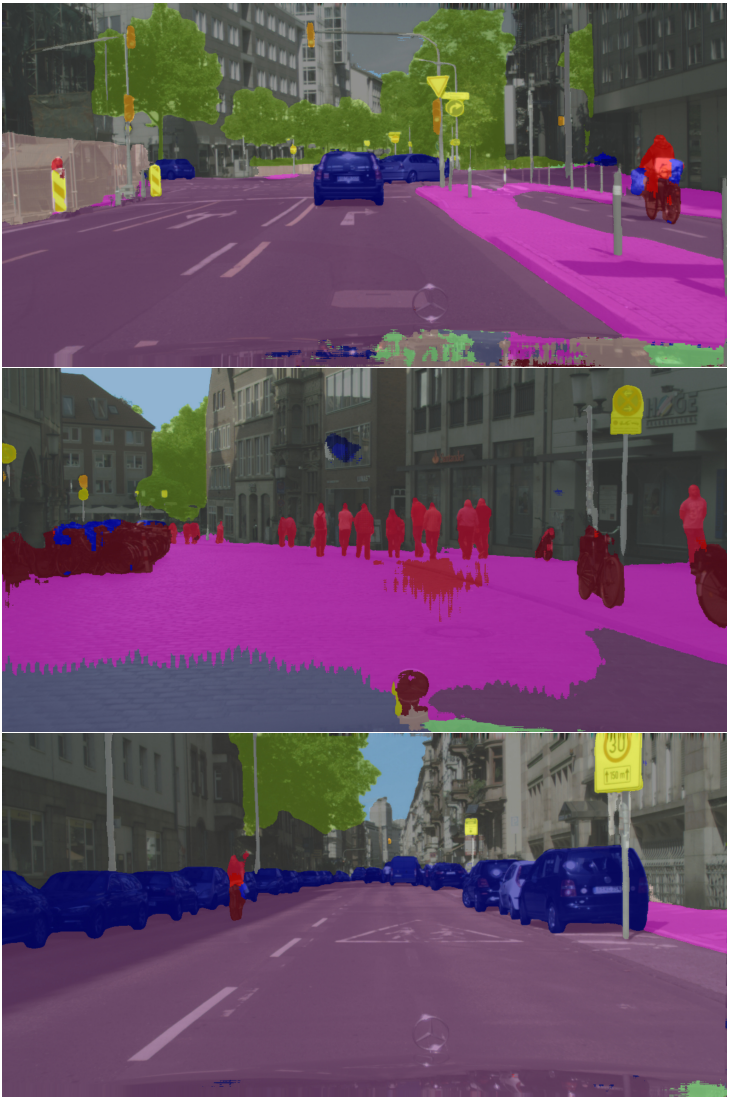


Figure 3.8: Qualitative results of the semantic segmentation task on the Cityscapes validation dataset. The CNN was trained on the BDD100K training dataset. This demonstrates the across-dataset performance. The corresponding color codes can be found in Figure 3.3.

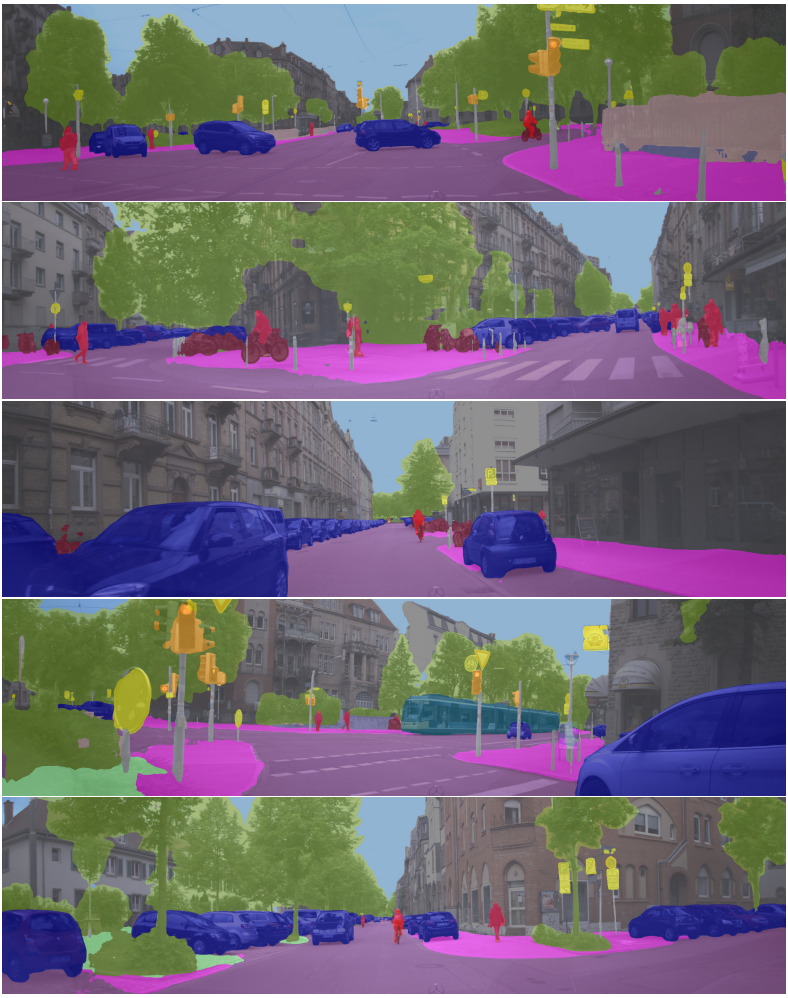


Figure 3.9: Qualitative results of the semantic segmentation task on images recorded with the research vehicle of MRT at KIT. The CNN was trained on the Cityscapes training dataset. The FoV of the camera in the research vehicle is more than twice as large as the FoV of the Cityscapes images. This demonstrates the performance across different camera characteristics. The corresponding color codes can be found in Figure 3.3.

	prior	baseline	MTL model
road	0.3293	0.977	0.973
sidewalk	0.0473	0.827	0.798
building	0.1917	0.920	0.902
wall	0.0064	0.584	0.478
fence	0.0072	0.618	0.559
pole	0.0129	0.639	0.562
traffic light	0.0017	0.654	0.636
traffic sign	0.0058	0.746	0.737
vegetation	0.1515	0.918	0.909
terrain	0.0073	0.614	0.615
sky	0.0293	0.941	0.928
person	0.0114	0.793	0.785
rider	0.0019	0.585	0.546
car	0.0570	0.944	0.939
truck	0.0026	0.788	0.766
bus	0.0034	0.862	0.837
train	0.0010	0.802	0.761
motorcycle	0.0007	0.464	0.435
bicycle	0.0062	0.735	0.713
mean	-	0.759	0.731

Table 3.1: IoU results for the semantic segmentation task on the Cityscapes dataset.

	prior	baseline	MTL model	baseline evaluated on Cityscapes
road	0.2160	0.949	0.907	0.945
sidewalk	0.0205	0.663	0.615	0.671
building	0.1489	0.859	0.827	0.881
wall	0.0036	0.308	0.281	0.263
fence	0.0081	0.453	0.476	0.343
pole	0.0097	0.550	0.449	0.510
traffic light	0.0014	0.593	0.454	0.465
traffic sign	0.0023	0.537	0.452	0.548
vegetation	0.1542	0.857	0.848	0.892
terrain	0.0091	0.506	0.423	0.428
sky	0.1791	0.948	0.927	0.910
person	0.0027	0.662	0.635	0.680
rider	0.0001	0.335	0.183	0.358
car	0.0907	0.903	0.900	0.920
truck	0.0101	0.510	0.552	0.469
bus	0.0063	0.791	0.762	0.535
train	0.0001	0.000	0.000	0.026
motorcycle	0.0002	0.424	0.441	0.192
bicycle	0.0002	0.404	0.478	0.508
mean	-	0.592	0.558	0.555

Table 3.2: IoU results for the semantic segmentation task (trained) on the BDD100K dataset.

	AP		LAMR	
	baseline	MTL model	baseline	MTL model
person	0.6368	0.6595	0.7071	0.6855
rider	0.5128	0.5455	0.5382	0.5201
car	0.7998	0.8111	0.5337	0.5466
truck	0.2819	0.3359	0.6955	0.6395
bus	0.5574	0.6580	0.4307	0.3381
train	0.4069	0.4622	0.5003	0.5051
motorcycle	0.3291	0.4478	0.6498	0.5448
bicycle	0.4894	0.5211	0.7307	0.7118
traffic light	0.5608	0.5893	0.6898	0.6648
traffic sign	0.5964	0.6164	0.7447	0.7274
mean	0.5171	0.5647	0.6221	0.5884

Table 3.3: Results for the object detection task on the Cityscapes dataset. The metrics are calculated based on an IoU threshold of 0.5.

	AP		LAMR	
	baseline	MTL model	baseline	MTL model
rider	0.4023	0.3291	0.5283	0.5760
car	0.8608	0.8589	0.4544	0.4473
truck	0.4560	0.4838	0.5994	0.5845
bus	0.6361	0.6245	0.3615	0.3734
motorcycle	0.3250	0.2340	0.5908	0.6070
bicycle	0.5543	0.4423	0.3785	0.4570
mean	0.5391	0.4954	0.4855	0.5075

Table 3.4: Results for the object detection task on the BDD100K dataset. The metrics are calculated based on an IoU threshold of 0.5.

of the semantic segmentation model on images from the research vehicle of MRT at KIT after training it on the Cityscapes dataset. After establishing the baselines, we evaluate our MTL model for simultaneous object detection and semantic segmentation.

Figure 3.4 and Figure 3.6 visualize the qualitative results of our MTL model for the semantic segmentation task on both datasets. Figure 3.5 and Figure 3.7 visualize the qualitative results for the object detection task.

Table 3.1 and Table 3.2 contain the results for the semantic segmentation task. On average, our MTL model performs approximately 3 percentage points worse than our baseline model on both datasets. One reason for the notable drop in performance might be that EfficientNet was designed to operate close to the network capacity limit: The width of each layer in the network is chosen to give the best accuracy to computation ratio for the image classification task. This ratio might not be optimal in a multi-task setting where not all computations can be shared. It might be possible to recover some of the lost accuracy without increasing the computation time too much by only scaling the network width. Answering if this is the optimal choice is however only possible by performing another grid search over the network hyperparameters from equation (2.24). This is computationally very expensive and not feasible for this thesis.

We observe that the results on the BDD100K dataset are worse than on the Cityscapes dataset. The reason for this is that the Cityscapes dataset is much cleaner than the BDD100K dataset. The former contains carefully selected images that were recorded in the centers of German cities. All recordings were performed during the day in dry weather conditions so that visibility is very good. The BDD100K dataset, on the other hand, contains images from dashcams that were recorded by many different drivers. The mounting position of the dashcam varies notably, and the dataset also contains recordings during the night, in rain, and in snow. This makes our perception tasks much more challenging. The BDD100K dataset is also much more imbalanced than the Cityscapes dataset. The worst AP values are achieved for the “train”, “motorcycle”, “bicycle”, and “rider” classes. These classes are underrepresented in the dataset and have a very small prior of 0.1 ‰ to 0.2 ‰.

We also want to show that our data pre-processing and augmentation techniques from Section 3.1.2 are effective to generalize across datasets. For this, we train our baseline model for semantic segmentation on the BDD100K dataset and then evaluate it on the Cityscapes dataset. The results can be found on the right column of Table 3.2. While they cannot match the results when evaluating on the same dataset, they come close and are only about 4 percentage points apart. A visual impression is provided in Figure 3.8. This figure also shows one of the failure cases of across-dataset training: Cobblestone streets, as present in the second image, are not common in the BDD100K dataset. The network therefore fails to make a good prediction. Our data augmentation techniques can help to compensate for different lighting and camera characteristics to some extent.

But they cannot change the underlying appearance of objects or infrastructure elements.

We employ our models on the self-driving research vehicle of MRT at KIT where they are part of the environment perception stack. Figure 3.9 shows some example images that were recorded close to MRT. The FoV of the camera in the research vehicle is more than twice as large as the FoV of the Cityscapes images which were used to train the corresponding model. This demonstrates the generalization across different camera characteristics.

The results for the object detection task are given in Table 3.3 and Table 3.4. On the Cityscapes dataset, our MTL model shows a notable increase in performance compared to the baseline model, measured both in the AP and LAMR metrics. But we also observe a decrease in performance of a similar size on the BDD100K dataset.

Our baseline models, as well as the MTL model, are comparably small, and therefore cannot quite achieve the accuracy of the best-performing models so far. The justification for this is that their small size makes them fast so that they can fulfill the real-time requirements of our application. We therefore also performed run-time measurements of our models on an *Nvidia Quadro RTX 8000* GPU in mixed-precision mode. All measurements were performed with a batch size of 1. This is because batching multiple images does not make sense for a sensor setup with a single camera. Images from multiple time steps would have to be combined in one batch before passing them to the GPU. The results for the older images would be of no interest once the batch is processed. Batching multiple images can however provide a large speedup, and can be effectively used with a multi-camera setup.

We measured an average run-time per image of 28.1 ms for the semantic segmentation baseline model, and 35.3 ms for the object detection baseline model. Running both networks side by side would result in a run-time of 63.4 ms. Our MTL model, on the other hand, only requires 38.2 ms on average to process one image. In other words, adding the semantic segmentation head to the object detection baseline network adds less than 10 % computational overhead. This is much faster than executing two individual networks. It also leaves a lot of room to scale up the network and recover the lost precision in the semantic segmentation task, if desired.

3.2 Non-Maxima Suppression with Feature Vectors

We showed in previous work that the assumptions of classical NMS fail in crowded scenes [Sal20a]. Such situations can occur in traffic scenes, e. g. when there are a lot of pedestrians. In our previous work, we proposed FeatureNMS to solve the problem and demonstrated that it performs better than classical NMS. It therefore makes sense to use FeatureNMS for our object detector. FeatureNMS relies on learning a similarity metric between bounding box detections. This similarity metric is used when the IoU between two detections does not allow to make a definite statement about whether they belong to the same object or not. In this thesis, we are also interested in learning a similarity metric for another reason: We will use it in Chapter 4.2 to re-identify objects between frames.

The FeatureNMS algorithm is given in Algorithm 1. The overall structure is similar to classical NMS. The detector first generates a set of proposals for the given input image. These are then sorted by the confidence scores in descending order. The following steps are then repeated until this sorted set of proposals is empty:

The proposal with the highest score is removed from the proposal set and added to the set of final detections. All remaining detections in the proposal set are then compared against this proposal and removed if they are duplicates. The duplicate detection heuristic is based on two metrics. The first one is the IoU between both bounding boxes. If it is smaller than a threshold t_{lower}, then the second detection is not a duplicate of the first. If it is, on the other hand, larger than a threshold t_{upper}, then the second detection is a duplicate. We choose $t_{\text{lower}} = 0.1$ and $t_{\text{upper}} = 0.9$ for our experiments. If the IoU is between t_{lower} and t_{upper}, it does not allow for a definite decision. In this case, the learned similarity metric is used. If it is below a threshold t_s, the second detection is a duplicate, and otherwise, it is not. The right threshold depends on the training objective of the similarity metric. We use $t_s = 1.2$.

In order to learn the similarity metric, we add another network head to our MTL model. It has the same structure as the network heads for object detection. This head produces an n-dimensional feature vector per anchor box. We choose $n = 32$ in our experiments. The similarity of two detections is then given by the

Algorithm 1 Proposed FeatureNMS algorithm. If the IoU between two bounding boxes does not allow to make a definite decision, we use the feature embedding similarity.

$\mathcal{P} \leftarrow$ getProposals($\mathbf{I}_{\text{image}}$)
$\mathcal{P} \leftarrow$ sort($\mathcal{P}$)
$\mathcal{D} \leftarrow \emptyset$
while $\mathcal{P} \neq \emptyset$ **do**
  $p \leftarrow$ pop($\mathcal{P}$)
  $isDuplicate \leftarrow$ false
  for $d \in \mathcal{D}$ **do**
    $iou \leftarrow$ getIoU(p, d)
    if $iou > t_{\text{upper}}$ **then**
      $isDuplicate \leftarrow$ true
    else if $iou > t_{\text{lower}}$ **then**
      $embeddingDistance \leftarrow$ getSimilarity(p, d)
      if $embeddingDistance < t_{\text{s}}$ **then**
        $isDuplicate \leftarrow$ true
      end if
    end if
  end for
  if $\neg isDuplicate$ **then**
    push($p, \mathcal{D}$)
  end if
end while

ℓ^2 distance of the corresponding feature vectors. The last layer of this network head is an ℓ^2 normalization layer. This ensures that all feature vectors have a length of one, i. e. they lie on a unit hypersphere. This is a common choice for metric learning, and a requirement for some loss functions like Margin based loss [Man+17].

3.2.1 Loss Function

The Margin based loss function suggests itself for learning the similarity metric. According to Manmatha et al. [Man+17], it achieves better results than other

loss functions for metric learning. But inspired by the idea of Focal Loss, we propose to replace the sampling step with an appropriate weighting of the loss function. This allows us to optimize for all possible pairs in each training step. At the same time, it still ensures that the loss is not dominated by easy negative examples. The proposed weighting function, that replaces sampling according to equation (2.22), is the following:

$$w_{\mathrm{WM}}(d) = \frac{\lambda}{\lambda + \exp\left(-n \cdot \left(d - \sqrt{2}\right)^2\right)} \tag{3.15}$$

Again, n is the dimensionality of the feature space. The parameter λ controls how much the easy negative examples are weighted down. We choose $\lambda = 0.2$ for our experiments.

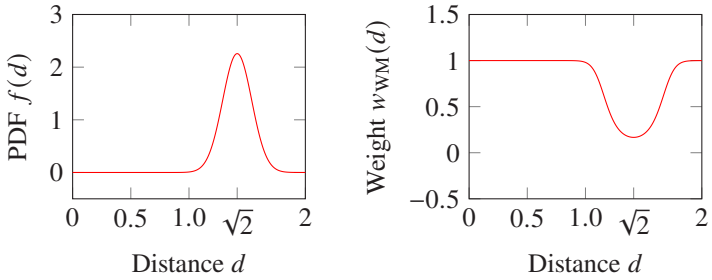
This weighting function ensures that the weight never gets larger than 1. In contrast, naive weighting by the inverse of the PDF in equation (2.22) would result in very large weighting factors for d close to zero. The complete loss function with this weighting factor is as follows:

$$\begin{aligned} \mathcal{L}_{w,\mathrm{WM}}(x_1, x_2, y_{1,2}) &= \begin{cases} \max\left(0, d_w(x_1, x_2) - b + m\right) & \text{for } y_{1,2} = 1 \\ \max\left(0, w \cdot (b - d_w(x_1, x_2) + m)\right) & \text{for } y_{1,2} = -1 \end{cases} \\ \text{where} \quad d_w(x_1, x_2) &= \|g_w(x_1) - g_w(x_2)\|_2 \\ w &= w_{\mathrm{WM}}(d_w(x_1, x_2)) \end{aligned} \tag{3.16}$$

The parameters b and m are the same as for the original Margin based loss. For our experiments, we follow the recommendation of choosing $b = 1.2$ and $m = 0.2$. Figure 3.10 visualizes the weighting effect of the proposed loss function for $n = 32$, i. e. for a feature space of $\mathbf{R}^{32}$.

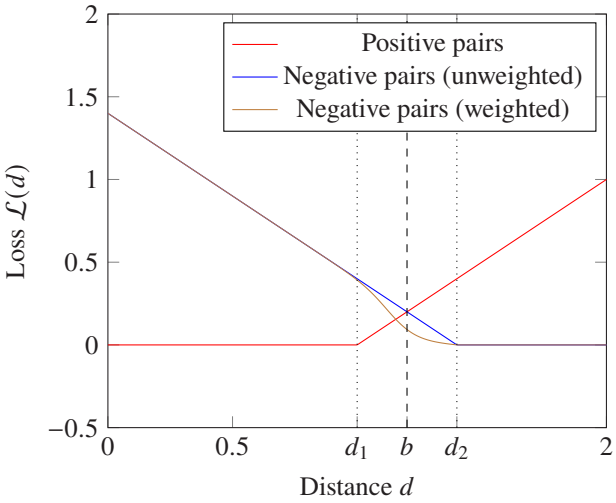
3.2.2 Evaluation

We already showed in previous work [Sal20a] that FeatureNMS can outperform classical NMS in crowded scenes. Here, we would like to investigate how well the approach works with our network architecture, and how much it helps in common driving scenes. For this, we first train our MTL model with the



(a) Approximate distribution of the pairwise distance between points that are uniformly distributed on a unit hypersphere in $\mathbf{R}^{32}$.

(b) Proposed weighting factor for negative training pairs (c. f. equation (3.15)).



(c) Comparison of the original Margin based loss and the proposed weighted margin based loss (c. f. equation (3.16)). We choose $\lambda = 0.2$, and follow the recommendations of $b = 1.2$ and $m = 0.2$. This results in $d_1 = b - m = 1.0$ and $d_2 = b + m = 1.4$ in the plot. Our weighted margin based loss weights down the original loss for the frequent but easy negative examples. This way, they do not dominate the total loss.

Figure 3.10: Visualization of the weighting effect of the proposed loss function for a feature space of $\mathbf{R}^{32}$.

additional network head that predicts the embedding vectors. We then compare its performance to the performance of the unmodified MTL model. In both cases, we use the classical NMS algorithm. This experiment shows the impact of the additional network head and the corresponding training objective on the network performance. We then evaluate the performance of the modified model with FeatureNMS.

The Cityscapes and BDD100K datasets, which we use in this thesis, cannot highlight the benefits of FeatureNMS. This is because the BDD100K dataset does not contain many crowded scenes. The Cityscapes dataset has crowded scenes, but it lacks instance annotations in the crowded areas. These areas are instead labeled as "crowd" and ignored during evaluation. It is nevertheless important to investigate how FeatureNMS performs on these datasets.

When training our MTL model with the additional network head that predicts the embedding vectors, we noticed that performance degraded a lot at first. We observed that the batch statistics have a high variance, especially in the last layers of the BiFPN. This means that the running averages of these statistics differ significantly from the individual batch statistics. It also means that the normalization performed by the Batch Normalization layers during inference is very different from the normalization during training.

This is the reason why we freeze the batch statistics after 1 500 000 training steps and train for another 300 000 steps with fixed statistics. This helps to recover most of the accuracy of our model. In theory, a better alternative would be to use a much larger batch size. This would reduce the variance of the batch statistics. But it would require hardware with enough memory to perform the training. Online Normalization, as used in our previous work, does not have this problem: It uses the running averages of the batch statistics also during training. It however slows down training significantly on our hardware. Another alternative is to use a normalization technique that does not depend on batch statistics, like Group Normalization.

The semantic segmentation results of our modified MTL model can be found in Table 3.5 for the Cityscapes dataset, and in Table 3.6 for the BDD100K dataset. The accuracy on the Cityscapes dataset decreases by 1.8 percentage points while the accuracy on the BDD100K dataset increases by 0.6 percentage points.

	without embedding	with embedding
road	0.973	0.967
sidewalk	0.798	0.790
building	0.902	0.890
wall	0.478	0.455
fence	0.559	0.509
pole	0.562	0.545
traffic light	0.636	0.630
traffic sign	0.737	0.718
vegetation	0.909	0.901
terrain	0.615	0.571
sky	0.928	0.901
person	0.785	0.772
rider	0.546	0.504
car	0.939	0.933
truck	0.766	0.737
bus	0.837	0.837
train	0.761	0.730
motorcycle	0.435	0.449
bicycle	0.713	0.707
mean	0.731	0.713

Table 3.5: IoU results for the semantic segmentation task on the Cityscapes dataset. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors.

	without embedding	with embedding
road	0.907	0.941
sidewalk	0.615	0.625
building	0.827	0.837
wall	0.281	0.264
fence	0.476	0.438
pole	0.449	0.431
traffic light	0.454	0.433
traffic sign	0.452	0.409
vegetation	0.848	0.853
terrain	0.423	0.448
sky	0.927	0.946
person	0.635	0.640
rider	0.183	0.241
car	0.900	0.906
truck	0.552	0.548
bus	0.762	0.754
train	0.000	0.000
motorcycle	0.441	0.503
bicycle	0.478	0.496
mean	0.558	0.564

Table 3.6: IoU results for the semantic segmentation task on the BDD100K dataset. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors.

	without embedding, classical NMS	with embedding, classical NMS	with embedding, FeatureNMS
person	0.6595	0.6430	0.6433
rider	0.5455	0.5283	0.5232
car	0.8111	0.8036	0.8055
truck	0.3359	0.3449	0.3519
bus	0.6580	0.7022	0.6957
train	0.4622	0.4364	0.4396
motorcycle	0.4478	0.4133	0.4220
bicycle	0.5211	0.5041	0.5039
traffic light	0.5893	0.5681	0.5613
traffic sign	0.6164	0.6030	0.5998
mean	0.5647	0.5547	0.5546

Table 3.7: AP results for the object detection task on the Cityscapes dataset based on an IoU threshold of 0.5. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors. It also compares the performance of classical NMS and FeatureNMS for the model that predicts the embedding vectors.

The object detection results on the Cityscapes dataset can be found in Table 3.7 and Table 3.8. The object detection results on the BDD100K dataset can be found in Table 3.9 and Table 3.10. On both datasets, the mean AP decreases and the mean LAMR increases a bit with our modified MTL model. We find that FeatureNMS performs slightly better than classical NMS, but it cannot recover the drop in accuracy caused by the additional network head. Our previous work suggests that FeatureNMS achieves notably better results in crowded scenes, but these cannot be evaluated on both datasets.

We continue with the modified MTL model from here on despite the drop in accuracy. The reason is that we want to use the same embedding vectors to track objects in Chapter 4.2. We also do not replace the Batch Normalization layers in this work to keep all results comparable.

	without embedding, classical NMS	with embedding, classical NMS	with embedding, FeatureNMS
person	0.6855	0.7074	0.7077
rider	0.5201	0.5409	0.5441
car	0.5466	0.5474	0.5486
truck	0.6395	0.6277	0.6184
bus	0.3381	0.2938	0.3007
train	0.5051	0.5012	0.4959
motorcycle	0.5448	0.5831	0.5747
bicycle	0.7118	0.7306	0.7306
traffic light	0.6648	0.6834	0.6838
traffic sign	0.7274	0.7377	0.7376
mean	0.5884	0.5953	0.5942

Table 3.8: LAMR results for the object detection task on the Cityscapes dataset based on an IoU threshold of 0.5. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors. It also compares the performance of classical NMS and FeatureNMS for the model that predicts the embedding vectors.

Adding the network head to predict the embedding vectors increases the run-time of our model from 38.2 ms to 41.9 ms.

	without embedding, classical NMS	with embedding, classical NMS	with embedding, FeatureNMS
rider	0.3291	0.3131	0.3348
car	0.8589	0.8477	0.8435
truck	0.4838	0.4144	0.4147
bus	0.6245	0.5645	0.5522
motorcycle	0.2340	0.2612	0.2760
bicycle	0.4423	0.4325	0.4292
mean	0.4954	0.4722	0.4751

Table 3.9: AP results for the object detection task on the BDD100K dataset based on an IoU threshold of 0.5. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors. It also compares the performance of classical NMS and FeatureNMS for the model that predicts the embedding vectors.

	without embedding, classical NMS	with embedding, classical NMS	with embedding, FeatureNMS
rider	0.5760	0.6057	0.5795
car	0.4473	0.5028	0.5035
truck	0.5845	0.6391	0.6403
bus	0.3734	0.4115	0.4237
motorcycle	0.6070	0.5723	0.5473
bicycle	0.4570	0.4774	0.4853
mean	0.5075	0.5348	0.5299

Table 3.10: LAMR results for the object detection task on the BDD100K dataset based on an IoU threshold of 0.5. This table compares the results of our MTL model with and without the network head that predicts the embedding vectors. It also compares the performance of classical NMS and FeatureNMS for the model that predicts the embedding vectors.

4 Multi-Frame Environment Perception

This chapter extends the single-frame model from the previous chapter to a multi-frame model. Section 4.1 presents the necessary modifications of the network structure. In the resulting network, two consecutive images of the video stream are used to estimate the displacement vectors of objects between frames. Section 4.2 presents a tracking approach based on the displacement estimates and on the similarity metric from FeatureNMS.

4.1 Optical Flow and Displacement Estimation

Chapter 3 introduced a neural network architecture that takes single images as input. The output is a list of bounding box detections and a semantic segmentation map. The goal of this section is to incorporate temporal information. Given two consecutive images, we want to estimate a displacement vector for each bounding box detection, as well as the change in width and height of the bounding box. Another view is that we estimate the velocities of all bounding boxes, assuming a fixed frame rate.

4.1.1 Network Structure

The neural network structure has to be modified so that it can take two input images. The naive approach would be to stack the two input images and use the result as input for the model. The backbone could learn to match patches of both images and use this information to estimate the bounding box displacement vectors. It is however possible that the displacement is large. This can happen, for example, if the camera rotates, or if a close object moves fast in the image plane. But if the displacement of an object is too large, its position in the first

frame might be outside of the receptive field of the convolutional layers at its position in the second frame. In this case, giving a good displacement estimate is impossible.

We therefore model a solution to this problem explicitly. The problem of estimating large displacements also arises when estimating optical flow. Here, good results can be achieved by using correlation layers which calculate the correlation between image patches or feature map patches. This approach is used by PWC-Net, which was presented in Section 2.3.4 and is one of the best performing CNN architectures for optical flow estimation.

We incorporate the optical flow estimation into our MTL model itself. Both input images are first processed individually by the backbone and part of the FPN in order to generate feature maps. The optical flow is estimated based on these feature maps. Afterwards, the feature maps of the first image are warped by the flow estimate. This aligns them with the feature maps of the second image. The flow estimate, the warped feature maps of the first image, and the feature maps of the second image are then concatenated and processed by the remaining network. Adding the flow estimate here is crucial: If the flow estimate is good, then it contains most of the information about the displacement. The warped feature maps of the first image and the feature maps of the second image would be nearly identical in this case—the displacement vectors cannot be recovered from them.

In single-shot inference scenarios, the proposed design doubles the computation time spent in the backbone. This is because most of the network is executed twice (once per input image). But in the context of self-driving cars, the network is evaluated on each frame of a continuous video stream. An image recorded at time step t is therefore used twice: It is considered as the "current frame" in time step t, and as the "previous frame" in time step $t + 1$. In both time steps, however, the image would be processed by the same backbone before reaching the optical flow estimation module. Therefore, these calculations do not have to be performed twice. Instead, the feature maps, which form the input to the flow estimation module, can be stored at time step t and reused at time step $t + 1$. This way, the backbone is only evaluated once per time step for the newly recorded image. Any remaining increase in computation time is then solely caused by the flow estimation module.

We perform the optical flow estimation with a PWC-Net style sub-network. This approach is efficient because it calculates the optical flow on a feature pyramid. Doing so allows to limit the search range on each pyramid level, and therefore reduces the computation time. But it also means that the flow estimation module has to be inserted into a part of the network where a feature pyramid is available. In our design, it is possible to insert it between the backbone and the BiFPN, after the BiFPN, or inside the BiFPN (between the repeated blocks). If it is inserted between the backbone and the BiFPN, the flow estimation module only has access to the backbone feature maps. This means that the feature maps at the lower pyramid levels have not yet been enriched with context information by the BiFPN. This, however, would be beneficial to get semantically correct matches. If the flow estimation module is inserted after the BiFPN, then no layers are left to perform a multi-scale fusion of the stacked feature maps. But this is important to guarantee a consistent flow estimation on all feature levels. We therefore insert the flow estimation module in the middle of the BiFPN.

The resulting architecture is visualized in Figure 4.1. The flow estimation module is depicted in Figure 4.3 and is similar to the one from PWC-Net (c. f. Figure 2.12). But we do not use a DenseNet style CNN sub-network or Leaky ReLU activation functions. Instead, our CNN block has the same structure as our network heads: One block consists of three depthwise separable convolutions, each followed by a Batch Normalization layer and a swish activation function. In contrast to PWC-Net, we also predict a mask $\mathbf{m}_l$ in each module. This mask allows the network to indicate which pixels have valid flow estimates. A pixel can have an invalid estimate if it is not observable in the other frame. Each module at feature level l takes the mask output $\mathbf{m}_{l+1}$ of the module at level $l+1$ as input. It is used to mask the pixels of the warped feature maps with invalid flow estimates.

The flow estimation module itself is embedded into a warping module. It aligns the input feature maps with each other according to the flow estimate and then fuses the information. This is visualized in Figure 4.2.

Our approach fuses the temporal information of two images by concatenating their warped feature maps. It is also possible to accumulate the information of many images in a Convolutional Long Short-Term Memory (ConvLSTM) cell instead. This might result in less noisy velocity estimates and more reliable detections of temporarily occluded objects. But it would increase the computational demands and memory requirements during training significantly.

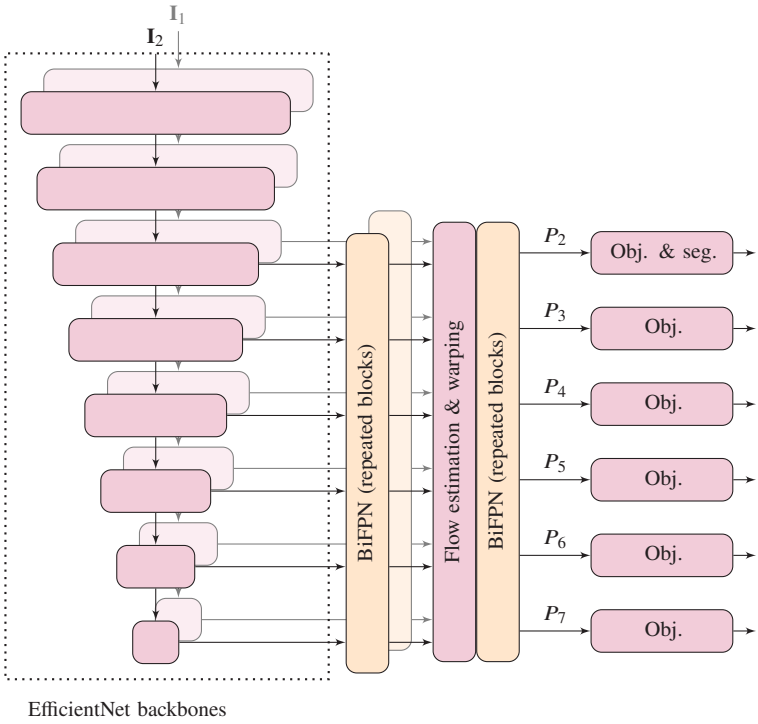


Figure 4.1: Overall structure of our proposed multi-frame model. The structure of the backbone and the BiFPN is the same as in Figure 3.2. The BiFPN is however cut in the middle, and the part before the cut is duplicated. One duplicate calculates feature maps for the first frame, and the other duplicate calculates feature maps for the second frame. These feature maps are combined by the flow estimation and warping module, which is visualized in Figure 4.2. The combined feature maps are then processed by the second half of the BiFPN and the network heads.

At this point, we therefore waive the evaluation of ConvLSTM cells for temporal information fusion.

In order to estimate the additional displacement parameters, we add another network head at each feature level. It estimates the four values $(t_{dx}, t_{dy}, t_{dw}, t_{dh})$

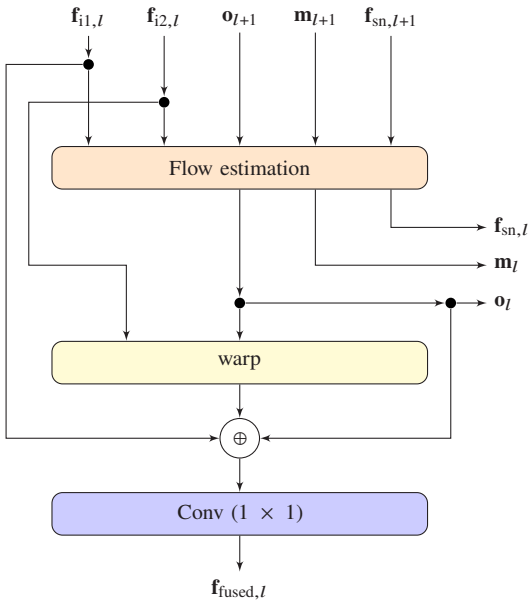


Figure 4.2: Overall structure of the flow estimation and warping module. It is instantiated once for every level in the feature pyramid. The flow estimation module itself is depicted in Figure 4.3. $\mathbf{f}_{\mathrm{i1},l}$ are the feature maps at level l extracted by the backbone for the current image. Analogously, $\mathbf{f}_{\mathrm{i2},l}$ are the feature maps at level l extracted by the backbone for the image from the previous time step. The output feature maps $\mathbf{f}_{\mathrm{fused},l}$ at level l are processed by the second half of the BiFPN and the network heads.

per anchor box. The values t_{dx} and t_{dy} are the x and y components of the displacement vector, measured in pixels. They are calculated as follows:

$$\begin{aligned} t_{dx} &= x_{c,\text{previous}} - x_{c,\text{current}} \\ t_{dy} &= y_{c,\text{previous}} - y_{c,\text{current}} \end{aligned} \tag{4.1}$$

Here, $(x_{c,\text{previous}}, y_{c,\text{previous}})$ is the center coordinate of the bounding box in the previous frame, and $(x_{c,\text{current}}, y_{c,\text{current}})$ is the center coordinate in the current frame.

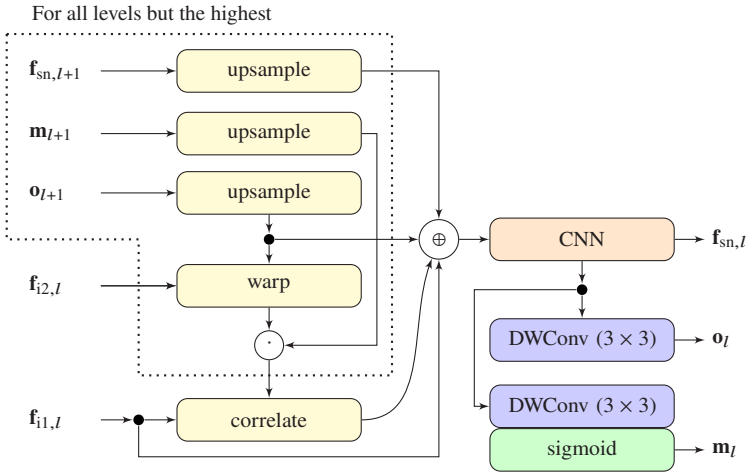


Figure 4.3: Structure of our proposed flow estimation module which is based on the PWC-Net module (c. f. Figure 2.12). But in contrast to PWC-Net, the CNN block has the standard structure of our network heads. It consists of three depthwise separable convolutions, each followed by a Batch Normalization layer and a swish activation function. We also predict a mask $\mathbf{m}_l$ at each feature level l. This mask indicate which pixels have valid flow estimates, and which have invalid flow estimates because they are not observable in the other frame. Each module at feature level l takes the mask output $\mathbf{m}_{l+1}$ of the module at level $l + 1$ as additional input. It is used to mask the pixels of the warped feature maps with invalid flow estimates.

The target values for the outputs that describe the change in size of the bounding box are calculated as follows:

$$\begin{aligned} t_{dw} &= \log\left(\frac{w_{\text{previous}}}{w_{\text{current}}}\right) \\ t_{dh} &= \log\left(\frac{h_{\text{previous}}}{h_{\text{current}}}\right) \end{aligned} \tag{4.2}$$

During inference, the size of a bounding box in the previous frame can then be estimated from the network outputs $(\tilde{t}_{dw}, \tilde{t}_{dh})$ and the estimate of the current size $(\tilde{w}_{\text{current}}, \tilde{h}_{\text{current}})$ as follows:

$$\begin{aligned} \tilde{w}_{\text{previous}} &= \tilde{w}_{\text{current}} \cdot \exp(\tilde{t}_{dw}) \\ \tilde{h}_{\text{previous}} &= \tilde{h}_{\text{current}} \cdot \exp(\tilde{t}_{dh}) \end{aligned} \tag{4.3}$$

4.1.2 Training Process

We train the existing network heads from Chapter 3 as before. The output of the additional network head is trained using smooth ℓ^1 loss. We use $\delta = 10$ for t_{dx} and t_{dy}, and $\delta = 0.1$ for t_{dw} and t_{dh}.

The flow estimation module gives estimates at each level of the feature pyramid (i. e. from level P_2 to P_7). These include both the upsampled estimates and the refined flow estimates at each level. We train the outputs of each of these modules using a supervised flow loss $\mathcal{L}_{\text{flow},l,\text{supervised}}$ and a self-supervised flow loss $\mathcal{L}_{\text{flow},l,\text{self-supervised}}$. Both contain a regularization term for the flow mask. In order to calculate the loss, we scale the predictions at each feature pyramid level to the original image size.

The supervised flow loss $\mathcal{L}_{\text{flow, supervised}}$ can only be used with datasets that have ground truth optical flow annotations. It is calculated as follows:

$$\mathcal{L}_{\text{flow},l,\text{supervised}}(\mathbf{o}_{\text{gt}}, \bar{\mathbf{o}}_l, \bar{\mathbf{m}}_l) = \bar{\mathbf{m}}_l \cdot \mathcal{L}_{\text{Huber},\,\delta=1}(\mathbf{o}_{\text{gt}}, \bar{\mathbf{o}}_l) + \mathcal{L}_{\text{mask}}(\bar{\mathbf{m}}_l) \tag{4.4}$$

Here, $\bar{\mathbf{m}}_l$ is the up-scaled mask, $\mathbf{o}_{\text{gt}}$ is the ground truth optical flow, and $\bar{\mathbf{o}}_l$ is the up-scaled predicted optical flow. $\mathcal{L}_{\text{mask}}$ is a regularization term for the mask. It is a weighted cross-entropy loss that forces the mask entries to be close to one:

$$\mathcal{L}_{\text{mask}}(\bar{\mathbf{m}}_l) = 0.2 \cdot \mathcal{L}_{\text{CE}}(\mathbf{1}, \bar{\mathbf{m}}_l) \tag{4.5}$$

The self-supervised flow loss $\mathcal{L}_{\text{flow,self-supervised}}$ can be calculated without optical flow annotations in the dataset. It is based on the photometric loss function

that Godard et al. use to learn monocular depth estimation in a self-supervised manner [God+19]:

$$\begin{aligned}
&\mathcal{L}_{\text{flow},l,\text{self-supervised}}(\mathbf{I}_1, \mathbf{I}_{2,\text{warped},l}, \bar{\mathbf{o}}_l, \bar{\mathbf{m}}_l) \\
&\quad = \bar{\mathbf{m}}_l \cdot \left(0.15 \cdot \mathcal{L}_{\ell^1}(\mathbf{I}_1, \mathbf{I}_{2,\text{warped},l}) + 0.85 \cdot \mathcal{L}_{\text{SSIM}}(\mathbf{I}_1, \mathbf{I}_{2,\text{warped},l})\right) \\
&\qquad + 0.1 \cdot \mathcal{L}_{\text{smooth}}(\mathbf{I}_1, \bar{\mathbf{o}}_l) \\
&\qquad + \mathcal{L}_{\text{mask}}(\bar{\mathbf{m}}_l)
\end{aligned} \tag{4.6}$$

Here, $\mathbf{I}_1$ is the first input image and $\mathbf{I}_{2,\text{warped},l}$ is the second input image, warped by the optical flow prediction $\mathbf{o}_l$ at feature level l. The term $\mathcal{L}_{\ell^1}(\mathbf{I}_1, \mathbf{I}_{2,\text{warped},l})$ ensures photometric consistency between the first image, and the second image warped by the flow estimate. It is the ℓ^1 loss on the raw pixel values. The term $\mathcal{L}_{\text{SSIM}}(\mathbf{I}_1, \mathbf{I}_{2,\text{warped},l})$ ensures structural similarity between the original and the warped image. It uses the SSIM metric proposed by Wang et al. which compares the local luminance, contrast, and structure of both images [Wan+04]:

$$\begin{aligned}
\mathcal{L}_{\text{SSIM}}(\mathbf{x}, \mathbf{y}) &= \min\left(\max\left(\frac{1 - SSIM(\mathbf{x}, \mathbf{y})}{2}, 1\right), 0\right) \\
SSIM(\mathbf{x}, \mathbf{y}) &= \frac{(2\mu_x\mu_y + C_1) \cdot (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1) \cdot (\sigma_x^2 + \sigma_y^2 + C_2)} \\
\mu_x &= \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i, \\
\sigma_x^2 &= \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2, \\
\sigma_y^2 &= \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \mu_y)^2, \\
\sigma_{xy} &= \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \\
\text{with } C_1 &= 0.0001, \quad C_2 = 0.0009
\end{aligned} \tag{4.7}$$

Here, N is the number of pixels in each of the two images.

The term $\mathcal{L}_{\text{smooth}}(\mathbf{I}_1, \bar{\mathbf{o}}_l)$ ensures the smoothness of the flow estimate. It punishes strong deviations between the flow estimates for neighboring pixels in

homogeneous image areas. This is done by multiplying the normalized spatial gradients of the flow map with an energy term based on the spatial gradients of the image:

$$\mathcal{L}_{\text{smooth}}(\mathbf{I}_1, \bar{\mathbf{o}}_l) = \frac{|\partial_x \bar{\mathbf{o}}_l|}{|\bar{\mathbf{o}}_l| + 1} \cdot \mathrm{e}^{-|\partial_x \mathbf{I}_1|} + \frac{\left|\partial_y \bar{\mathbf{o}}_l\right|}{|\bar{\mathbf{o}}_l| + 1} \cdot \mathrm{e}^{-|\partial_y \mathbf{I}_1|} \tag{4.8}$$

The normalization of the flow gradients by the absolute value of the flow is important to avoid shrinking of the flow estimates.

In theory, our whole model can be trained end-to-end. Even though we do not have one dataset that contains all annotations, it is possible to interleave the data and mix data from different datasets in one batch. For each training example, the outputs without annotations would be masked in the loss calculation.

We found however that end-to-end training of our model is unstable in practice. In the beginning, the flow estimates are inaccurate. This means that the warping of the feature maps is inconsistent. This, in turn, makes it difficult for the network heads to discover meaningful patterns. The training progress is dominated by the improvements in flow estimation.

At some point, the flow estimates become more accurate, and the warped feature maps become more meaningful. The network heads start to learn, but at this point, they still have high errors in their outputs. This causes large gradients, and as a consequence the learned convolutional filters in the backbone change notably. This, in turn, reduces the accuracy of the flow estimation again.

We circumvent this instability by executing the training of the model in three stages:

1. We train the single-frame model from Chapter 3 on single images. We then freeze all trainable parameters of the backbone and the first half of the BiFPN. The idea is that the frozen parts of the model already learned to extract all relevant features for object detection and pixel-wise semantic segmentation. If they still contain fine details with accurate localization information then it should also be possible to estimate displacements based on these features. This is a reasonable assumption because localization information is a requirement for high accuracy on the pixel-wise semantic segmentation task.

2. We remove the second half of the BiFPN as well as the network heads and add the flow estimation module. We then train the resulting model on the FlyingChairs dataset [Dos+15a]. It is important to keep everything but the flow estimation model frozen. The model would otherwise adapt to the characteristics of the synthetic images and forget about the original task. Freezing the backbone can unfortunately also reduce the flow estimation performance because the synthetic training images are out of domain.

3. We also freeze the flow estimation module and add the remaining parts of the final model, i. e. the second half of the BiFPN and the network heads. These are then trained to perform the final tasks.

In each of the above stages, we train for 1 800 000 steps in total and follow the training protocol that was presented at the beginning of Chapter 3. On our GPUs, it takes around 3 months to perform all training stages of the multi-frame model.

We also tried to unfreeze the whole multi-frame network in the last training stage, after performing the first 300 000 training iterations to re-initialize the network heads. We used the self-supervised loss from equation (4.6) to train the flow estimation module and keep its output from degenerating. We however found that this gives worse results for the flow estimation than when keeping the earlier layers frozen throughout the last training stage.

4.1.3 Evaluation

Figure 4.4 contains qualitative results of the optical flow estimation on the Cityscapes dataset. As a comparison, Figure 4.5 contains qualitative results that were obtained with the classical optical flow estimation algorithm of Farnebäck [Far03]. The direction of movement in the image plane is encoded in the hue of the color, while the magnitude is encoded in its brightness (or value). The moving objects in the displayed scenes are clearly visible, and the estimated directions and magnitudes are consistent.

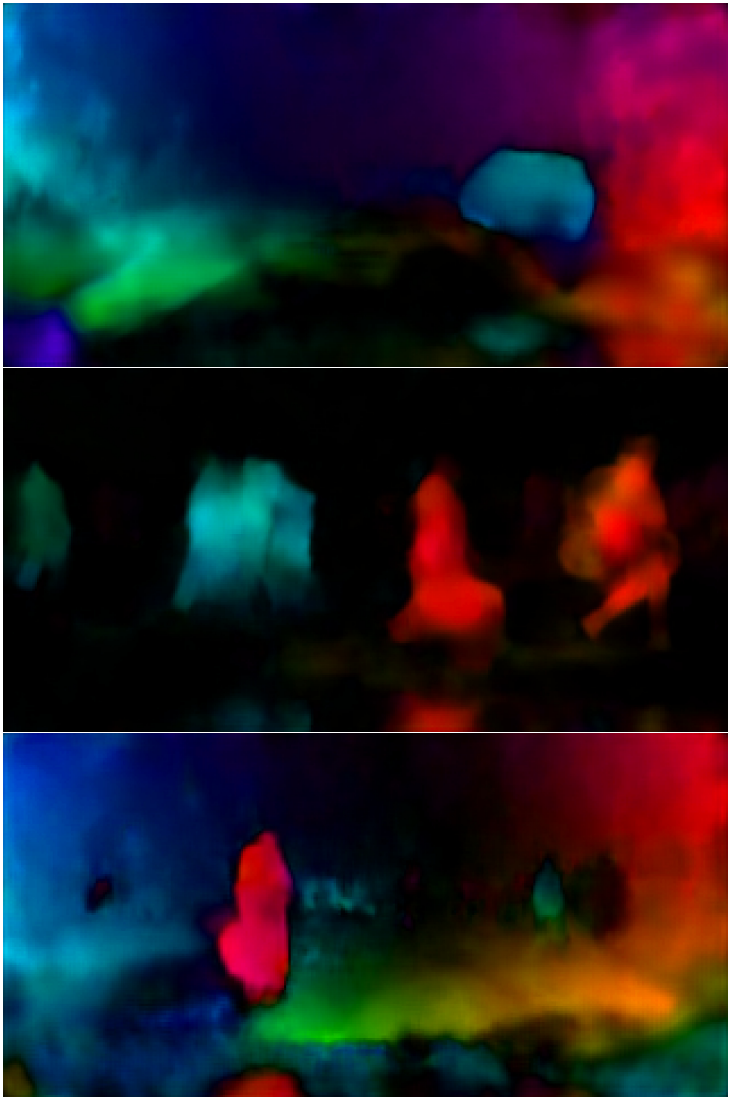


Figure 4.4: Example outputs of the optical flow estimation module at the P_2 feature level on the Cityscapes validation dataset. The direction of movement in the image plane is encoded in the hue of the color, while the magnitude is encoded in its brightness.

Figure 4.5: Classical optical flow estimates computed with the algorithm of Farnebäck [Far03]. The input images from the Cityscapes validation dataset were resized to match the size of the P_2 feature maps. The direction of movement in the image plane is encoded in the hue of the color, while the magnitude is encoded in its brightness.

	single-frame	multi-frame
road	0.941	0.937
sidewalk	0.625	0.606
building	0.837	0.832
wall	0.264	0.240
fence	0.438	0.440
pole	0.431	0.421
traffic light	0.433	0.423
traffic sign	0.409	0.406
vegetation	0.853	0.851
terrain	0.448	0.431
sky	0.946	0.944
person	0.640	0.634
rider	0.241	0.238
car	0.906	0.904
truck	0.548	0.543
bus	0.754	0.747
train	0.000	0.000
motorcycle	0.503	0.422
bicycle	0.496	0.452
mean	0.564	0.551

Table 4.1: IoU results for the semantic segmentation task on the BDD100K dataset. This table compares the results of our single-frame and our multi-frame model.

	AP		LAMR	
	single-frame	multi-frame	single-frame	multi-frame
rider	0.3348	0.3665	0.5795	0.5549
car	0.8435	0.8536	0.5035	0.4771
truck	0.4147	0.4503	0.6403	0.6063
bus	0.5522	0.5998	0.4237	0.3996
motorcycle	0.2760	0.3457	0.5473	0.5335
bicycle	0.4292	0.3275	0.4853	0.5188
mean	0.4751	0.4906	0.5299	0.5150

Table 4.2: Results for the object detection task on the BDD100K dataset. The metrics are calculated based on an IoU threshold of 0.5. This table compares the results of our single-frame and our multi-frame model.

	MAE for $\tilde{t}_{dx}$		MAE for $\tilde{t}_{dy}$	
	from CNN	constant	from CNN	constant
rider	5.6924	15.9447	2.4106	2.6946
car	5.0279	15.3317	2.5472	4.1594
truck	5.3155	10.7355	3.3733	4.1994
bus	7.2913	13.6860	3.9427	4.5850
motorcycle	4.3395	20.0551	4.7324	4.7228
bicycle	4.8784	18.0543	2.3677	3.9360
mean	5.4242	15.6345	3.2290	4.0495

Table 4.3: MAE between the ground truth displacement (t_{dx}, t_{dy}), and the corresponding estimates $(\tilde{t}_{dx}, \tilde{t}_{dy})$. The table compares the estimates from our CNN architecture to estimates obtained with a constant position assumption.

	MAE for $\exp(\tilde{t}_{dw})$		MAE for $\exp(\tilde{t}_{dh})$	
	from CNN	constant	from CNN	constant
rider	0.1699	0.0977	0.1616	0.1116
car	0.1616	0.1115	0.1352	0.0673
truck	0.1379	0.0663	0.1426	0.0583
bus	0.1378	0.0620	0.1332	0.0528
motorcycle	0.2435	0.1467	0.2199	0.1277
bicycle	0.1553	0.1077	0.1248	0.0516
mean	0.1677	0.0987	0.1529	0.0782

Table 4.4: MAE between the ground truth size ratios $(\exp(t_{dw}), \exp(t_{dh}))$, and the corresponding estimates $(\exp(\tilde{t}_{dw}), \exp(\tilde{t}_{dh}))$. The table compares the estimates from our CNN architecture to estimates obtained with a constant size assumption.

In this chapter, we however only provide quantitative results on the BDD100K dataset. The reason for this is that the Cityscapes dataset only contains single-frame annotations. This means that the network head for the displacement estimates cannot be trained with this dataset.

We first compare the semantic segmentation and object detection performance of our multi-frame architecture to our single-frame architecture. There are multiple factors that can influence the performance: The multi-frame model has access to the information from two frames. This could increase accuracy in some situations, e. g. when an object is not clearly visible in one of the two frames. On the other hand, the multi-frame model was not trained end-to-end which can reduce accuracy. Finally, the additional network head for the displacement estimation can also have an effect on accuracy.

Table 4.1 contains the results of our experiments for the semantic segmentation task. Table 4.2 contains our results for the object detection task. The accuracy of our multi-frame model is slightly reduced on the semantic segmentation task compared to our single-frame model. It however shows slightly better results on the object detection task, both in the AP and the LAMR metrics. Overall, the performance of both models is comparable.

Table 4.3 and Table 4.4 contain the results of the displacement estimation. The results from our multi-frame model are compared against a baseline with a constant position / constant size assumption. It can be clearly seen that our multi-frame model successfully exploits the temporal information: The displacement estimates of our model are much better than the baseline. Our model reduces the mean absolute error from 15.6 pixels to 5.4 pixels in x-direction, and from 4.0 to 3.2 pixels in y-direction. When it comes to the bounding box size, however, the baseline with a constant size assumption performs very well. The results show that the bounding box sizes do not change much between consecutive frames on average. Our model performs slightly worse than the baseline and cannot provide additional information. We therefore propose to only estimate the change in position for tracking but assume that the bounding box size is constant.

We measure a run-time of 131.2 ms for our multi-frame model. For this, we only pass the current frame through the backbone and the first half of the BiFPN and reuse the extracted feature maps of the previous frame. This run-time is much higher than the 41.9 ms that we measured for our single-frame model.

It should however be noted that most of the additional computation time is spent in the correlation cost layers. In comparison to most other layers, the implementation of the correlation cost layer from Tensorflow Addons is much less optimized. Also, it only supports 32 bit floating point inputs and outputs. On the *Nvidia Quadro RTX 8000* GPU, computations with this data type are much slower than with 16 bit floating point or 8 bit integer values. This suggests that much higher inference speeds can be achieved with an optimized implementation of the correlation cost layer. This is however not the focus of this thesis.

4.2 Tracking Approach

Detecting objects and estimating their velocities is not enough for self-driving cars. The future trajectories and behavior of objects can only be accurately predicted if their past behavior is known. For this, the objects must have stable identifiers, i. e. the detections have to be associated over time to create tracks.

In this section, we present a tracking approach based on our CNN architecture from the previous sections. It follows our previous work [Sal21]. We propose a simple tracking-by-detection architecture that consists of a motion model and an association heuristic. The latter takes position and appearance information into account.

4.2.1 Motion Model

We use a constant velocity motion model with a Kalman filter, and track objects in the image domain. Since our target frame rate is 10 fps or higher, the time between consecutive frames is not more than 100 ms. The constant velocity assumption seems reasonable for this time span since the maximum acceleration of objects is limited. For longer prediction horizons, the constant velocity assumption however is obviously violated. The filter therefore must not trust the system model too much so that it can also follow accelerated objects.

We do not take the ego-motion into account since it is not available in all datasets, and since we do not have 3D information about the scene. Doing so could however improve the predictions of the motion model: Especially changes in the yaw rate of the ego vehicle can cause violations of the constant velocity assumption for tracked objects.

The state vector of our Kalman filter is $(p_x, p_y, p_w, p_h, v_x, v_y, v_w, v_h)^T$. Here, p_x and p_y are the center coordinates of the object bounding box in the image domain. The width and height of the bounding box are given by p_w and p_h respectively. We denote the corresponding velocities as v_x, v_y, v_w and v_h. The state transition matrix of the Kalman filter is then given by:

$$\mathbf{F}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

Here, dt is the time between two consecutive frames and has to be set to match the frame rate of the camera. The parameters p_x, p_y, p_w and p_h are directly observable. The multi-frame model from the previous section can also directly observe v_x, v_y, v_w and v_h with the displacement estimation network head. The single-frame model from the previous chapter, however, cannot observe any velocities. In both cases, the observation matrix $\mathbf{H}_k$ is trivial.

Apart from the state transition and observation models, we need to determine the covariance matrices of the process noise $\mathbf{Q}_k$ and of the observation noise $\mathbf{R}_k$. These are often tuned manually so that the filter tracks the target reasonably well. In this work, we however learn the optimal parameters using gradient descent by following the work of Abbeel et al. [Abb+05]. To generate training data, we first run our CNN on a training dataset for the Kalman filter. Then, we generate the training annotations by manually assigning the detections to ground truth bounding boxes. Afterwards, we train the Kalman filter with gradient descent. In each training step, the detections from the CNN are fed to the Kalman filter as measurements. The prediction error is calculated based on the corresponding ground truth annotations, and the parameters of the Kalman filter are updated based on the gradients.

4.2.2 Assignment Problem and Track Management

A precise motion model is important. But it is even more important to solve the assignment problem accurately. Given a set of existing tracks and a new measurement, we have to decide if the new measurement corresponds to any of the existing tracks, and if so to which. We use several features that provide evidence for this assignment problem:

- The similarity metric that we presented in Section 3.2 for FeatureNMS. It contains appearance information and can be used to associate objects based on their visual features. This is very informative especially after longer occlusions or in cases where the assumptions of the motion model are violated.

  Each track t stores the feature vectors $\mathbf{f}_{t,i}$ of the last n corresponding detections. We chose $n = 10$ for our experiments. The distance between the new measurement with feature vector $\mathbf{f}_o$ and track t is then the

minimum distance between the track feature vectors and the measurement feature vector:

$$d_{\text{feature},t,o} = \min_i \|\mathbf{f}_{t,i} - \mathbf{f}_o\|_2 \tag{4.10}$$

- The displacement vector that is predicted by the model from Section 4.1. Given a bounding box in the current video frame and its displacement vector estimate, the bounding box position in the previous frame can be estimated by summing them up. This directly follows from equation (4.1). We can then calculate the ℓ^2 distance between the last position from the track $\mathbf{p}_{\text{previous},t}$, and the estimated bounding box position in the previous frame $\tilde{\mathbf{p}}_{\text{previous},o}$:

$$d_{\text{displacement},t,o} = \|\mathbf{p}_{\text{previous},t} - \tilde{\mathbf{p}}_{\text{previous},o}\|_2 \tag{4.11}$$

- The distance between the new measurement and the state of the Kalman filter after the prediction step. This indicates how likely it is to observe the new measurement given the filter state. We use the Mahalanobis distance for this:

$$d_{\text{state},t,o} = \sqrt{(\mathbf{x_o} - \mathbf{x_t})^T \mathbf{S}_t^{-1} (\mathbf{x_o} - \mathbf{x_t})} \tag{4.12}$$

  Here, $\mathbf{x_o}$ is the measurement in the current frame. The vector $\mathbf{x_t}$ is the state of the Kalman filter after the prediction step to the current frame, projected into the measurement space. The matrix $\mathbf{S}_t$ is the corresponding covariance matrix.

- A class distance that punishes the assignment of new detections to tracks of another class. We chose the distance to be 0 if the class of the new detection matches the class of the existing track, and 1 otherwise:

$$d_{\text{class},t,o} = \begin{cases} 0 & \text{if } c_t = c_o \\ 1 & \text{otherwise} \end{cases} \tag{4.13}$$

  This is a very crude heuristic. It can be improved by having different class distances based on how likely it is to confuse the classes. For example, the distance between the "tram" and the "pedestrian" classes can be chosen larger than the distance between the "bicycle" and "motorbike"

classes, because it seems less likely to mix up the former. For simplicity, we however refrain from determining different costs per class pair in our experiments.

Based on these features, we train a classifier to decide if a measurement belongs to a given track or not. This is done using a linear SVM [CV95]. This way, we only have to calculate a weighted sum of the presented distances during inference, and compare it against a threshold:

$$\begin{aligned} d_{\text{total},t,o} = & w_1 \cdot d_{\text{feature},t,o} + \\ & w_2 \cdot d_{\text{displacement},t,o} + \\ & w_3 \cdot d_{\text{state},t,o} + \\ & w_4 \cdot d_{\text{class},t,o} + \\ & b \end{aligned} \tag{4.14}$$

Here, w_i are weights and b is a bias term which allows to fix the threshold at zero. We obtain the training data for the SVM from the same dataset that we already annotated to learn the parameters of the Kalman filter.

If the presented distances would allow for a completely unambiguous association, and if the SVM classifier was perfect, then the problem would be solved with this step. In practice, however, it can happen that the SVM classifier predicts that multiple detections belong to an existing track, or that a detection belongs to multiple existing tracks. We therefore use the Hungarian algorithm [Kuh55] to find the minimum cost assignment. We then threshold the costs and obtain the final pairwise assignments.

After this step, there can be unassigned existing tracks or detections. We deal with these by applying two track management heuristics. Each unassigned detection creates a new track. This track is however not yet visible in the output of the tracking module. The reason for this is that we want to suppress false positives. We only consider the track to be stable after n_{stable} consecutive observations. Once it is stable, it will be visible in the output of the tracking module. We chose $n_{\text{stable}} = 3$ for our experiments. This value suppresses most false positive detections while keeping the detection delay reasonably low at our target frame rate.

The second heuristic concerns the deletion of stale tracks. We delete a track if it did not receive any new observations within a time span of t_{stale}. We chose $t_{\text{stale}} = 0.5\,s$. This time is long enough to suppress most false negatives from the detector.

It is however not long enough to handle full occlusions of objects in most cases. But we argue that also our motion model is not suitable for handling predictions of typical time horizons of full occlusions. If tracking of fully occluded objects is a requirement, then we propose to rely on a different motion model for long-term predictions. This prediction model must take the behavior of the object into account, and perform ego-motion compensation. The association after the long-term occlusion can then be made based on this long-term prediction and the appearance feature vector.

In summary, we execute the following steps for each new video frame:

1. Run the CNN on the input image.
2. Execute the prediction step of the Kalman filter to get state estimates for the current time step for all existing tracks.
3. Calculate the individual distances and the final distance according to equation (4.14) for each pair of existing tracks and new detections.
4. Solve the assignment problem using the Hungarian algorithm [Kuh55] and threshold the maximum costs.
5. Execute the update step of the Kalman filter for each successfully assigned track to incorporate the new measurement.
6. Create new tracks for unassigned detections. These are however invisible to the downstream users of the tracker until they have at least $n_{\text{stable}} = 3$ consecutive observations.
7. Delete tracks without observations during the last $t_{\text{stale}} = 0.5\,s$.

4.2.3 Evaluation

As before, we evaluate our proposed approach on the BDD100K tracking dataset. In particular, we want to answer the question if and how much our

additional assignment metrics improve the object tracking performance. For this, we establish a baseline that uses the Mahalanobis distance $d_{\text{state},t,o}$ between the Kalman filter states and the new detections, and the class distance $d_{\text{class},t,o}$. These metrics can be calculated for any object detector, and do not rely on our proposed architectures. We then compare the baseline to several other combinations of metrics. These combinations include our proposed embedding distance $d_{\text{feature},t,o}$, or the distance $d_{\text{displacement},t,o}$ based on our displacement estimates. The results for all experiments can be found in Table 4.5.

Experiment 1 establishes our baseline. It already achieves a low MOTP value, i. e. a small localization error. There are two reasons for this: The first is that our object detector has good localization accuracy. The second is that our tracking approach relies on frequent detections of the tracked object. Both the localization error of the object detector and the accuracy of the motion model contribute to the overall localization error. The contribution of the motion model however becomes larger when its prediction horizon increases. But long prediction horizons only occur when an object is not detected in several consecutive frames, or when the new detections cannot be associated.

Even though the MOTP of our baseline is good, it is still the highest amongst all of our experiments. The baseline also has the highest miss ratio r_{m} of all. Since we do not use any of our additional metrics, fewer new detections can be associated with existing tracks. The Kalman filter therefore receives fewer updates and has to predict longer time horizons. The baseline in experiment 1 achieves a good MOTA metric, also compared to our other experiments. Despite the comparably high miss ratio r_{m}, it exhibits a low false positive ratio r_{fp}, and a low mismatch ratio r_{mme}.

Experiments 2, 4, and 6 add the displacement based distance $d_{\text{displacement},t,o}$ to the assignment step of the corresponding experiments 1, 3, and 5. These experiments therefore require a detection network that can estimate displacement vectors, like our architecture from Chapter 4.1. The additional metric reduces the miss ratio r_{m} notably in experiments 2 and 6. This also results in a significantly improved MOTA metric—experiment 2 achieves the highest MOTA value of all. The additional information provided by the displacement vector allows to make more informed assignments of new detections to existing tracks. This also means that longer tracks can be formed on average. The share of mostly tracked and partially tracked objects increases and the share of mostly lost objects decreases. In experiment 6, the information from the displacement

#	assignment metrics	MOTP	MOTA	r_{m}	r_{fp}	r_{mme}
1	C, KF	0.207	0.370	0.459	0.144	0.026
2	C, KF, D	0.206	0.391	0.439	0.145	0.025
3	C, KF, E	0.202	0.368	0.417	0.160	0.054
4	C, KF, D, E	0.202	0.365	0.419	0.161	0.055
5	C, E	0.202	0.335	0.427	0.161	0.077
6	C, D, E	0.202	0.354	0.422	0.157	0.068

(a) Object tracking metrics as described in Section 2.5.3. Here, r_{m} is the miss ratio, r_{fp} is the false positive ratio, and r_{mme} is the mismatch ratio.

#	assignment metrics	mostly tracked	partially tracked	mostly lost
1	C, KF	14.5 %	45.5 %	39.9 %
2	C, KF, D	16.9 %	45.9 %	37.1 %
3	C, KF, E	22.2 %	46.2 %	31.5 %
4	C, KF, D, E	21.9 %	46.3 %	31.8 %
5	C, E	21.3 %	46.9 %	31.7 %
6	C, D, E	21.9 %	46.4 %	31.7 %

(b) Ratio of successfully tracked objects, depending on the total object lifespan. We consider an object to be mostly tracked if it is tracked for at least 80 % of its lifespan, and to be mostly lost if it is tracked for less than 20 % of its lifespan. Otherwise, we consider the object to be partially tracked.

Table 4.5: Tracking results of our approach for all object classes on the BDD100K tracking dataset. This table compares the results when using different combinations of metrics in the assignment step. In these tables, we use the following abbreviations for the assignment metrics: C: Use class distance $d_{\text{class},t,o}$. KF: Use the Mahalanobis distance $d_{\text{state},t,o}$ between the Kalman filter state and the new detection. D: Use displacement vector based distance $d_{\text{displacement},t,o}$. E: Use the embedding vector distance $d_{\text{feature},t,o}$.

vector also introduces a geometric constraint which is completely missing in experiment 5. This reduces the false positive ratio r_{fp} and the mismatch ratio r_{mme}. Interestingly, it seems that the information from the displacement vector is not useful in presence of all other metrics. Experiment 4, which combines all of our metrics in the assignment step, therefore does not show an improvement compared to experiment 3. This will shortly be discussed in more detail.

Experiments 3 and 4 add the embedding vector distance $d_{\text{feature},t,o}$ to the assignment costs of the corresponding experiments 1 and 2. They therefore require a detection network that predicts embedding vectors, like the one proposed in Chapter 3.2. The additional metric results in many more successful associations of new detections to existing tracks. The embedding vectors allow to associate objects based on their visual appearance, and therefore allow associations when the other metrics fail, e. g. after short occlusions. This reduces the miss ratio r_{m} significantly and also improves the localization accuracy as given by the MOTP metric. The share of mostly tracked and partially tracked objects increases and the share of mostly lost objects decreases. Trusting more in appearance features and less in geometric constrains however does not only result in more successful associations, but also in more identity switches. The mismatch ratio r_{mme} increases notably, and therefore the MOTA metric drops.

Because of this observation, and because of the drop in accuracy that we observed in Chapter 3.2 when learning the embedding vectors, we want to evaluate the discriminativeness of the embedding vectors for the tracking task. Experiment 5 therefore waives all geometric constraints, and only relies on the embedding vector distance $d_{\text{feature},t,o}$ and the class distance $d_{\text{class},t,o}$ in the association step. This means giving up a lot of valuable information, and as a consequence, this experiment has the highest mismatch ratio r_{mme} and the worst MOTA result. However, the achieved performance is still close to the other experiments and the achieved mismatch ratio r_{mme} is still good. This shows that the learned embedding vectors are quite discriminative. Even more discriminative embedding vectors at the cost of higher computation time might be obtained by choosing a higher dimensionality of the embedding space. Experiment 6 adds the displacement vector based distance $d_{\text{displacement},t,o}$ to experiment 5, and thereby also introduces a geometric constraint. This constraint however is not posed in the current frame, but only in the previous frame. It

recovers most of the performance loss from experiment 3 to 5, but still performs slightly worse.

We already mentioned that experiment 4, which combines all of our metrics in the association step, does not perform better than experiment 3, which uses all but the displacement vector based metric. It seems that the displacement vector based metric does not provide additional information when there is already geometric information from the Kalman filter and appearance information from the embedding vector. One possible explanation is that both the embedding vector and the displacement vector are after all calculated based on the same visual information. Both might therefore be correlated and have similar failure cases.

Based on our experiments, the best results are either achieved by combining $d_{\text{class},t,o}$, $d_{\text{state},t,o}$ and $d_{\text{displacement},t,o}$, or $d_{\text{class},t,o}$, $d_{\text{state},t,o}$ and $d_{\text{feature},t,o}$. The first combination should be chosen if a high MOTA value is desired. The second combination should be chosen if a low miss ratio is desired. Combining all metrics does not yield further improvements in our experiments. This statement however does not necessarily generalize to modified of completely different CNN architectures—the combination of all four metrics in the association step might still be beneficial for these. It must also be reevaluated when replacing the Batch Normalization layers in our architecture, which we suggested in Chapter 3.2 as a possible solution to the drop in accuracy when learning the embedding vectors.

5 Conclusions and Outlook

In this work, we presented an efficient and accurate approach to vision-based object detection and tracking, as well as pixel-wise semantic segmentation. It is designed to meet the requirements of self-driving cars and is based on deep learning.

Our approach is based on existing work that we introduced in Chapter 2. In particular, we base our work on EfficientNet, EfficientDet, and PWC-Net. EfficientNet is a computationally very efficient CNN backbone for image classification. EfficientDet is a single-stage object detector that was designed around the EfficientNet backbone. PWC-Net finally is a CNN architecture for optical flow estimation.

Chapter 3.1 introduced our CNN architecture for simultaneous object detection and semantic segmentation. This MTL model is based on EfficientNet and EfficientDet which show an outstanding accuracy to computation time ratio and achieve state of the art results. Our proposed architecture is scalable, which means that it is easy to trade computation time for accuracy. In our experiments, we confirmed that our proposed architecture achieves good results on the challenging Cityscapes and BDD100K datasets. At the same time, it has a very low inference time and can meet the computational constraints and real-time requirements of the application. Our data augmentation techniques result in good generalization across datasets and across camera characteristics. This allows us to successfully employ our approach inside the software stack of the self-driving research vehicle of MRT at KIT.

In Chapter 3.2, we proposed to predict embedding vectors for each bounding box detection. This requires only one additional network head in our CNN architecture. We presented a modified version of Margin loss that does not depend on sampling but instead weights down the easy negative examples. The embedding vectors are discriminative and allow to associate bounding box detections. We use them for FeatureNMS as well as for object tracking.

FeatureNMS is a novel NMS approach that we initially proposed in an earlier work. There, we show that it can improve recall in crowded scenes where the assumptions of classical NMS do not hold.

In Chapter 4.1, we proposed a CNN architecture that takes two consecutive frames as input. Feature maps are extracted for each frame individually and then fed into an optical flow estimation network. The feature maps are aligned based on the flow estimate and then forwarded to the network heads. This allows us to estimate displacement vectors between both frames for all detected objects. We use this information for object tracking.

Chapter 4.2 finally introduced our tracking approach. It is computationally very cheap and based on simple techniques. We use a constant velocity motion model with a Kalman filter, the Hungarian algorithm to solve the assignment problem, and an SVM classifier to learn the optimal weighting of assignment costs. We showed that the evaluation metrics of our tracking approach improve significantly when adding costs based on the embedding vectors or the displacement vectors to the assignment costs. The tracking approach relies on a strong object detector that can extract these additional features. It was designed to be used together with the CNN architectures from the previous chapters. The combination of both achieves good object tracking results and has proven useful for real-world usage inside the research vehicle of MRT at KIT.

In Chapter 3.2, we noticed that the combination of our model architecture, a small batch size, Batch Normalization, and our proposed loss for feature embedding learning leads to degraded accuracy. This should be investigated in future work. Our architecture can be retrained with a larger batch size on hardware that has enough memory for this. Alternatively, Batch Normalization can be exchanged for another normalization approach that does not rely on batch statistics.

Future work should also answer the question if the compound scaling parameters of EfficientNet and EfficientDet are also optimal for our MTL model. This requires a grid search which is very computationally expensive. Also, the run-time of the multi-frame model should be measured again with a more efficient implementation of the correlation layer.

Finally, this work focused on perception in the image domain and did not consider the estimation of 3D world coordinates. These are however important

for many tasks in a self-driving car and are also necessary for many sensor fusion approaches. There are many possibilities to obtain 3D world coordinates for the object detections from our approach. Rough estimates can be easily obtained with a fixed object size assumption per class and with a ground plane assumption. Another alternative is to associate the 2D detections with 3D detections from another sensor or to add depth information from disparity estimates or projected lidar points to the 2D detections.

Instead of performing post-processing steps like these, it is also possible to directly regress 3D world coordinates inside our CNN architecture. This is even possible without any modification of the architecture itself. In this case, the network learns to estimate the depth of objects from a single image. It is also possible to add depth estimates as an additional layer to the input images so that the network can reuse this information. These depth estimates can e. g. come from a stereo matching approach or a lidar sensor. An alternative is to perform stereo matching inside a modified version of our model itself. This is conceptually similar to our proposed module for optical flow estimation. It is then necessary to provide input images from a stereo camera pair.

Bibliography

[Aba+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv: 1603.04467 [cs.DC]. URL: http://arxiv.org/abs/1603.04467 (visited on 12/31/2020).

[Abb+05] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y. Ng, and Sebastian Thrun. "Discriminative Training of Kalman Filters". In: *Robotics: Science and Systems I*. Ed. by Sebastian Thrun, Gaurav S. Sukhatme, and Stefan Schaal. Cambridge, MA, USA: The MIT Press, 2005, pp. 289–296. DOI: 10.15607/RSS.2005.I.038.

[Bay+17] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18.153 (2017), 153:1–153:43.

[BC13] Asad A. Butt and Robert T. Collins. "Multi-target Tracking by Lagrangian Relaxation to Min-cost Network Flow". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Portland, OR, USA: IEEE Computer Society, 2013, pp. 1846–1853. DOI: 10.1109/CVPR.2013.241.

[Ber+16] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. "Fully-Convolutional Siamese Networks for Object Tracking". In: *Computer Vision - ECCV 2016 Workshops*. Ed. by Gang Hua and Hervé Jégou. Amsterdam, The Netherlands: Springer International Publishing, 2016, pp. 850–865. DOI: 10.1007/978-3-319-48881-3_56.

[Bew+16] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Tozeto Ramos, and Ben Upcroft. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. Phoenix, AZ, USA: IEEE, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495. DOI: 10.1109/TPAMI.2016.2644615.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450 (visited on 12/31/2020).

[BML19] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. "Tracking Without Bells and Whistles". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Ed. by IEEE. Seoul, South Korea, 2019, pp. 941–951. DOI: 10.1109/ICCV.2019.00103.

[Bol+10] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. "Visual object tracking using adaptive correlation filters". In: *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2010*. San Francisco, CA, USA: IEEE Computer Society, 2010, pp. 2544–2550. DOI: https://doi.org/10.1109/CVPR.2010.5539960.

[Bro+93] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature Verification Using a Siamese Time Delay Neural Network". In: *Advances in Neural Information Processing Systems 6, (7th NIPS Conference)*. Ed. by Jack D. Cowan, Gerald Tesauro, and Joshua Alspector. Denver, CO, USA: Morgan Kaufmann, 1993, pp. 737–744.

[BS08] Keni Bernardin and Rainer Stiefelhagen. "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics". In: *EURASIP Journal on Image and Video Processing* (2008), pp. 1–10. DOI: 10.1155/2008/246309.

[Car+20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872 (visited on 12/31/2020).

[Car97] Rich Caruana. "Multitask Learning". In: *Machine Learning* 28.1 (1997), pp. 41–75. DOI: 10.1023/A:1007379606734.

[Che+09] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. "Large Scale Online Learning of Image Similarity through Ranking". In: *Pattern Recognition and Image Analysis, 4th Iberian Conference, IbPRIA 2009*. Ed. by Helder Araújo, Ana Maria Mendonça, Armando J. Pinho, and M. Inés Torres. Póvoa de Varzim, Portugal: Springer, 2009, pp. 11–14. DOI: 10.1007/978-3-642-02172-5_2.

[Che+14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*. 2014. arXiv: 1412.7062 [cs.CV]. URL: https://arxiv.org/abs/1412.7062 (visited on 12/31/2020).

[Che+17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. arXiv: 1706.05587 [cs.CV]. URL: http://arxiv.org/abs/1706.05587 (visited on 12/31/2020).

[Che+18a] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2018), pp. 834–848. DOI: 10.1109/TPAMI.2017.2699184.

[Che+18b] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *Computer Vision - ECCV 2018 - 15th European Conference*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu,

and Yair Weiss. Munich, Germany: Springer, 2018, pp. 833–851. DOI: 10.1007/978-3-030-01234-2_49.

[Che+18c] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks". In: *Proceedings of the 35th International Conference on Machine Learning (ICML) 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Stockholmsmässan, Stockholm, Sweden: PMLR, 2018, pp. 793–802.

[Chi+19] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Köster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. "Online Normalization for Training Neural Networks". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems (NeurIPS) 2019*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. Vancouver, BC, Canada: Curran Associates, Inc., 2019, pp. 8431–8441.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a Similarity Metric Discriminatively, with Application to Face Verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. San Diego, CA, USA: IEEE Computer Society, 2005, pp. 539–546. DOI: 10.1109/CVPR.2005.202.

[Cho17] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 1800–1807. DOI: 10.1109/CVPR.2017.195.

[Cir+12] Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems (NIPS) 2012*. Lake Tahoe, NV, USA: Curran Associates, Inc., 2012, pp. 2852–2860.

[Cor+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 3213–3223. DOI: 10.1109/CVPR.2016.350.

[Cos+17] Huseyin Coskun, Felix Achilles, Robert S. DiPietro, Nassir Navab, and Federico Tombari. "Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization". In: *IEEE International Conference on Computer Vision (ICCV) 2017*. Venice, Italy: IEEE Computer Society, 2017, pp. 5525–5533. DOI: 10.1109/ICCV.2017.589.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[Cyb89] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. DOI: 10.1007/BF02551274.

[CZH18] Han Cai, Ligeng Zhu, and Song Han. *ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware*. 2018. arXiv: 1812.00332 [cs.LG]. URL: https://arxiv.org/abs/1812.00332 (visited on 12/31/2020).

[Dan+15] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg. "Convolutional Features for Correlation Filter Based Visual Tracking". In: *2015 IEEE International Conference on Computer Vision (ICCV) Workshop*. Santiago, Chile: IEEE Computer Society, 2015, pp. 621–629. DOI: 10.1109/ICCVW.2015.84.

[DHS10] John C. Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *COLT 2010 - The 23rd Conference on Learning Theory*. Ed. by Adam Tauman Kalai and Mehryar Mohri. Haifa, Israel: Omnipress, 2010, pp. 257–269.

[Dol+17] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. "Pedestrian Detection: An Evaluation of the State of the Art". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.4 (2017), pp. 743–761. DOI: 10.1109/TPAMI.2011.155.

[Dos+15a] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE Computer Society, 2015, pp. 2758–2766. DOI: 10.1109/ICCV.2015.316.

[Dos+15b] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE Computer Society, 2015, pp. 2758–2766. DOI: 10.1109/ICCV.2015.316.

[DS18] Xingping Dong and Jianbing Shen. "Triplet Loss in Siamese Network for Object Tracking". In: *Computer Vision - ECCV 2018 - 15th European Conference*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Munich, Germany: Springer, 2018, pp. 472–488. DOI: 10.1007/978-3-030-01261-8_28.

[DT05] Navneet Dalal and Bill Triggs. "Histograms of Oriented Gradients for Human Detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. San Diego, CA, USA: IEEE Computer Society, 2005, pp. 886–893. DOI: 10.1109/CVPR.2005.177.

[Dua+19] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. "CenterNet: Keypoint Triplets for Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, South Korea: IEEE, 2019, pp. 6568–6577. DOI: 10.1109/ICCV.2019.00667.

[Eve+12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012)*. 2012. URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html (visited on 12/31/2020).

[Eve+15] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (2015), pp. 98–136. DOI: 10.1007/s11263-014-0733-5.

[Far03] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Image Analysis, 13th Scandinavian Conference, SCIA 2003*. Ed. by Josef Bigün and Tomas Gustavsson. Halmstad, Sweden: Springer, 2003, pp. 363–370. DOI: 10.1007/3-540-45103-X_50.

[Fel+10] Pedro F. Felzenszwalb, Ross B. Girshick, David A. McAllester, and Deva Ramanan. "Object Detection with Discriminatively Trained Part-Based Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167.

[FPZ17] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. "Detect to Track and Track to Detect". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 3057–3065. DOI: 10.1109/ICCV.2017.330.

[Gho+17] Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. "Learning Human Motion Models for Long-Term Predictions". In: *2017 International Conference on 3D Vision (3DV)*. Qingdao, China: IEEE Computer Society, 2017, pp. 458–466. DOI: 10.1109/3DV.2017.00059.

[Gir+14] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, USA: IEEE Computer Society, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[Gir15] Ross B. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE Computer Society, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Providence, RI, USA: IEEE Computer Society, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[God+19] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. *Digging Into Self-Supervised Monocular Depth Estimation*. 2019. arXiv: 1806.01260 [cs.CV]. URL: https://arxiv.org/abs/1806.01260 (visited on 12/31/2020).

[Gon+19] Ting Gong, Tyler Lee, Cory Stephenson, Venkata Renduchintala, Suchismita Padhy, Anthony Ndirango, Gokce Keskin, and Oguz H. Elibol. “A Comparison of Loss Weighting Strategies for Multi task Learning in Deep Neural Networks”. In: *IEEE Access* 7 (2019), pp. 141627–141632. DOI: 10.1109/ACCESS.2019.2943604.

[HCL06] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. New York, NY, USA: IEEE Computer Society, 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[He+16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[He+16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity Mappings in Deep Residual Networks”. In: *Computer Vision - ECCV 2016 - 14th European Conference*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Amsterdam, The Netherlands: Springer, 2016, pp. 630–645. DOI: 10.1007/978-3-319-46493-0_38.

[He+17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. "Mask R-CNN". In: *IEEE International Conference on Computer Vision (ICCV) 2017*. Venice, Italy: IEEE Computer Society, 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[He+18] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. "A Twofold Siamese Network for Real-Time Object Tracking". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 4834–4843. DOI: 10.1109/CVPR.2018.00508.

[Hen+15] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. "High-Speed Tracking with Kernelized Correlation Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), pp. 583–596. DOI: 10.1109/TPAMI.2014.2345390.

[Ho95] Tin Kam Ho. "Random decision forests". In: *Third International Conference on Document Analysis and Recognition (ICDAR) 1995*. Montreal, Canada: IEEE Computer Society, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994.

[Hor91] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: 10.1016/0893-6080(91)90009-T.

[How+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV]. URL: https://arxiv.org/abs/1704.04861 (visited on 12/31/2020).

[How+19] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, et al. "Searching for MobileNetV3". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, South Korea: IEEE, 2019, pp. 1314–1324. DOI: 10.1109/ICCV.2019.00140.

[HS81] Berthold K. P. Horn and Brian G. Schunck. "Determining Optical Flow". In: *Artificial Intelligence* 17.1-3 (1981), pp. 185–203. DOI: 10.1016/0004-3702(81)90024-2.

[HSS18] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-Excitation Networks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

[Ilg+17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 1647–1655. DOI: 10.1109/CVPR.2017.179.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Ed. by Francis R. Bach and David M. Blei. Lille, France: JMLR.org, 2015, pp. 448–456.

[Kal60] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45.

[KB15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2015. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980 (visited on 12/31/2020).

[KGC18] Alex Kendall, Yarin Gal, and Roberto Cipolla. “Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 7482–7491. DOI: 10.1109/CVPR.2018.00781.

[Kir+19] Alexander Kirillov, Ross B. Girshick, Kaiming He, and Piotr Dollár. “Panoptic Feature Pyramid Networks”. In: *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: Computer Vision Foundation / IEEE, 2019, pp. 6399–6408. DOI: 10.1109/CVPR.2019.00656.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems (NeurIPS) 2012*. Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. Lake Tahoe, NV, USA: Curran Associates, Inc., 2012, pp. 1106–1114.

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[KZS15] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition". In: *ICML Deep Learning Workshop 2015*. Lille, France, 2015.

[Lab19] Marie Labrie. *NVIDIA Introduces DRIVE AGX Orin—Advanced, Software-Defined Platform for Autonomous Machine*. 2019. URL: https://nvidianews.nvidia.com/_gallery/download_pdf/5df9a4d4ed6ae535592da3d8/ (visited on 12/31/2020).

[Law+19] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. *CornerNet-Lite: Efficient Keypoint Based Object Detection*. 2019. arXiv: 1904.08900 [cs.CV]. URL: https://arxiv.org/abs/1904.08900 (visited on 12/31/2020).

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. arXiv: 1312.4400 [cs.NE]. URL: https://arxiv.org/abs/1312.4400 (visited on 12/31/2020).

[LD18] Hei Law and Jia Deng. "CornerNet: Detecting Objects as Paired Keypoints". In: *Computer Vision - ECCV 2018 - 15th European Conference*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Munich, Germany: Springer, 2018, pp. 765–781. DOI: 10.1007/978-3-030-01264-9_45.

[LeC+89] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[LH19] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101 (visited on 12/31/2020).

[Li+17] Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. *WebVision Database: Visual Learning and Understanding from Web Data*. 2017. arXiv: 1708.02862 [cs.CV]. URL: http://arxiv.org/abs/1708.02862 (visited on 12/31/2020).

[Li+18] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. "High Performance Visual Tracking With Siamese Region Proposal Network". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 8971–8980. DOI: 10.1109/CVPR.2018.00935.

[Lin+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context". In: *Computer Vision - ECCV 2014 - 13th European Conference*. Ed. by David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars. Zurich, Switzerland: Springer, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[Lin+17a] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D. Reid. "RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 5168–5177. DOI: 10.1109/CVPR.2017.549.

[Lin+17b] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. "Feature Pyramid Networks for Object Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[Lin+17c] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *IEEE International Conference on Computer Vision (ICCV) 2017*.

Venice, Italy: IEEE Computer Society, 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[Liu+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Computer Vision - ECCV 2016 - 14th European Conference*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Amsterdam, The Netherlands: Springer, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

[Liu+18] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. "Path Aggregation Network for Instance Segmentation". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 8759–8768. DOI: 10.1109/CVPR.2018.00913.

[LJD19] Shikun Liu, Edward Johns, and Andrew J. Davison. "End-To-End Multi-Task Learning With Attention". In: *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: Computer Vision Foundation / IEEE, 2019, pp. 1871–1880. DOI: 10.1109/CVPR.2019.00197.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE Computer Society, 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[LYU18] Wenjie Luo, Bin Yang, and Raquel Urtasun. "Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting With a Single Convolutional Net". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 3569–3577. DOI: 10.1109/CVPR.2018.00376.

[Man+17] R. Manmatha, Chao-Yuan Wu, Alexander J. Smola, and Philipp Krähenbühl. "Sampling Matters in Deep Embedding Learning". In: *IEEE International Conference on Computer Vision (ICCV) 2017*. Venice, Italy: IEEE Computer Society, 2017, pp. 2859–2867. DOI: 10.1109/ICCV.2017.309.

[MBR17] Julieta Martinez, Michael J. Black, and Javier Romero. “On Human Motion Prediction Using Recurrent Neural Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 4674–4683. DOI: 10.1109/CVPR.2017.497.

[Mey+18] Annika Meyer, Niels Ole Salscheider, Piotr Franciszek Orzechowski, and Christoph Stiller. “Deep Semantic Lane Segmentation for Mapless Driving”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain: IEEE, 2018, pp. 869–875. DOI: 10.1109/IROS.2018.8594450.

[MP69] M. Minsky and S. A. Papert. *Perceptrons*. Cambridge, MA, USA: MIT Press, 1969.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE Computer Society, 2015, pp. 1520–1528. DOI: 10.1109/ICCV.2015.178.

[OPH94] Timo Ojala, Matti Pietikäinen, and David Harwood. “Performance evaluation of texture measures with classification based on Kullback discrimination of distributions”. In: *12th IAPR International Conference on Pattern Recognition, Conference A: Computer Vision & Image Processing (ICPR) 1994*. Jerusalem, Israel: IEEE, 1994, pp. 582–585. DOI: 10.1109/ICPR.1994.576366.

[PRF11] Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. “Globally-optimal greedy algorithms for tracking a variable number of objects”. In: *The 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2011*. Colorado Springs, CO, USA: IEEE Computer Society, 2011, pp. 1201–1208. DOI: 10.1109/CVPR.2011.5995604.

[RB17] Anurag Ranjan and Michael J. Black. “Optical Flow Estimation Using a Spatial Pyramid Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 2720–2729. DOI: 10.1109/CVPR.2017.291.

[Red+16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[Ren+15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NeurIPS) 2015*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett. Montreal, Quebec, Canada: Curran Associates, Inc., 2015, pp. 91–99.

[RF17] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[RF18] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV]. URL: http://arxiv.org/abs/1804.02767 (visited on 12/31/2020).

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi. Munich, Germany: Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[RHW86a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[RHW86b] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[Ros57] Frank Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton (Project Para)*. Buffalo, NY, USA: Cornell Aeronautical Laboratory, 1957.

[RT16] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. "Recurrent Instance Segmentation". In: *Computer Vision - ECCV 2016 - 14th European Conference*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Amsterdam, The Netherlands: Springer, 2016, pp. 312–329. DOI: 10.1007/978-3-319-46466-4_19.

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE]. URL: https://arxiv.org/abs/1710.05941 (visited on 12/31/2020).

[Sal20a] Niels Ole Salscheider. "FeatureNMS: Non-Maximum Suppression by Learning Feature Embeddings". In: *25th International Conference on Pattern Recognition, ICPR 2020*. Virtual Event / Milan, Italy: IEEE, 2020, pp. 7848–7854. DOI: 10.1109/ICPR48806.2021.9412930.

[Sal21] Niels Ole Salscheider. *Object Tracking by Detection with Visual and Motion Cues*. 2021. arXiv: 2101.07549 [cs.CV]. URL: https://arxiv.org/abs/2101.07549 (visited on 01/20/2021).

[San+18a] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[San+18b] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. “How Does Batch Normalization Help Optimization?” In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS) 2018*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Montréal, Canada: Curran Associates, Inc., 2018, pp. 2488–2498.

[SAN16] Russell Stewart, Mykhaylo Andriluka, and Andrew Y. Ng. “End-to-End People Detection in Crowded Scenes”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 2325–2333. DOI: https://doi.org/10.1109/CVPR.2016.255.

[SCZ08] Florian Schroff, Antonio Criminisi, and Andrew Zisserman. “Object Class Segmentation using Random Forests”. In: *Proceedings of the British Machine Vision Conference 2008*. Ed. by Mark Everingham, Chris J. Needham, and Roberto Fraile. Leeds, UK: British Machine Vision Association, 2008, pp. 1–10. DOI: 10.5244/C.22.54.

[SK18] Ozan Sener and Vladlen Koltun. “Multi-Task Learning as Multi-Objective Optimization”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS) 2018*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Montréal, Canada: Curran Associates, Inc., 2018, pp. 525–536.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE Computer Society, 2015, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682.

[SL18] Niels Ole Salscheider and Martin Lauer. “Instance Segmentation by Learning Pixel Neighbour Relations with a CNN”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Ed. by Wei-Bin Zhang, Alexandre M. Bayen, Javier J. Sánchez Medina, and Matthew J. Barth. Maui, Hi, USA: IEEE, 2018, pp. 3463–3468. DOI: 10.1109/ITSC.2018.8569291.

[Son+16] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. "Deep Metric Learning via Lifted Structured Feature Embedding". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 4004–4012. DOI: 10.1109/CVPR.2016.434.

[Sun+18] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018, pp. 8934–8943. DOI: 10.1109/CVPR.2018.00931.

[SZ14] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV]. URL: http://arxiv.org/abs/1409.1556 (visited on 12/31/2020).

[Sze+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[Sze+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[Sze+17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. Ed. by Satinder P. Singh and Shaul Markovitch. San Francisco, CA, USA: AAAI Press, 2017, pp. 4278–4284.

[Tan+19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *2019 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA: Computer Vision Foundation / IEEE, 2019, pp. 2820–2828. DOI: 10.1109/CVPR.2019.00293.

[TH12] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012. URL: www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf (visited on 02/08/2021).

[Tia+19] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. "FCOS: Fully Convolutional One-Stage Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, South Korea: IEEE, 2019, pp. 9626–9635. DOI: 10.1109/ICCV.2019.00972.

[TL19] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Long Beach, CA, USA: PMLR, 2019, pp. 6105–6114.

[TPL20] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE Computer Society, 2020, pp. 10778–10787. DOI: 10.1109/CVPR42600.2020.01079.

[Uij+13] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: 1607.08022 [cs.CV]. URL: https://arxiv.org/abs/1607.08022 (visited on 12/31/2020).

[Vin+16] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NeurIPS) 2016*. Ed. by Daniel D. Lee, Masashi

Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. Barcelona, Spain: Curran Associates, Inc., 2016, pp. 3630–3638.

[VJ01] Paul A. Viola and Michael J. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Kauai, HI, USA: IEEE Computer Society, 2001, pp. 511–518. DOI: 10.1109/CVPR.2001.990517.

[Wan+04] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[Wan+13] Xiaoyu Wang, Ming Yang, Shenghuo Zhu, and Yuanqing Lin. "Regionlets for Generic Object Detection". In: *IEEE International Conference on Computer Vision (ICCV) 2013*. Sydney, Australia: IEEE Computer Society, 2013, pp. 17–24. DOI: 10.1109/ICCV.2013.10.

[Wan+14] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. "Learning Fine-Grained Image Similarity with Deep Ranking". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, USA: IEEE Computer Society, 2014, pp. 1386–1393. DOI: 10.1109/CVPR.2014.180.

[WBP17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE International Conference on Image Processing (ICIP)*. Beijing, China: IEEE, 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[WH18] Yuxin Wu and Kaiming He. "Group Normalization". In: *Computer Vision - ECCV 2018 - 15th European Conference*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Munich, Germany: Springer, 2018, pp. 3–19. DOI: 10.1007/978-3-030-01261-8_1.

[WSH19] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. "Wider or Deeper: Revisiting the ResNet Model for Visual Recognition". In: *Pattern Recognition* 90 (2019), pp. 119–133. DOI: 10.1016/j.patcog.2019.01.006.

[Xie+17] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 5987–5995. DOI: 10.1109/CVPR.2017.634.

[Yan+18] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications". In: *Computer Vision - ECCV 2018 - 15th European Conference*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Munich, Germany: Springer, 2018, pp. 289–304. DOI: 10.1007/978-3-030-01249-6_18.

[Yan+19] Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. "RepPoints: Point Set Representation for Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, South Korea: IEEE, 2019, pp. 9656–9665. DOI: 10.1109/ICCV.2019.00975.

[YHD16] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. "Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness". In: *Computer Vision - ECCV 2016 Workshops*. Ed. by Gang Hua and Hervé Jégou. Amsterdam, The Netherlands: Springer International Publishing, 2016, pp. 3–10. DOI: 10.1007/978-3-319-49409-8_1.

[Yu+20] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE Computer Society, 2020, pp. 2633–2642. DOI: 10.1109/CVPR42600.2020.00271.

[Zha+17] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid Scene Parsing Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE Computer Society, 2017, pp. 6230–6239. DOI: 10.1109/CVPR.2017.660.

[ZL16] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2016. arXiv: 1611.01578 [cs.LG]. URL: https://arxiv.org/abs/1611.01578 (visited on 12/31/2020).

[ZLN08] Li Zhang, Yuan Li, and Ramakant Nevatia. “Global data association for multi-object tracking using network flows”. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Anchorage, AK, USA: IEEE Computer Society, 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587584.

[ZWK19] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. *Objects as Points*. 2019. arXiv: 1904.07850 [cs.CV]. URL: http://arxiv.org/abs/1904.07850 (visited on 12/31/2020).

[ZZK19] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. “Bottom-Up Object Detection by Grouping Extreme and Center Points”. In: *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: Computer Vision Foundation / IEEE, 2019, pp. 850–859. DOI: 10.1109/CVPR.2019.00094.

Own Publications

Journal Articles

[1] Wei Tian, Niels Ole Salscheider, Yunxiao Shan, Long Chen, and Martin Lauer. "A Collaborative Visual Tracking Architecture for Correlation Filter and Convolutional Neural Network Learning". In: *IEEE Transactions on Intelligent Transportation Systems* 21.8 (2020), pp. 3423–3435. DOI: 10.1109/TITS.2019.2928963.

[2] Ömer Sahin Tas, Niels Ole Salscheider, Fabian Poggenhans, Sascha Wirges, Claudio Bandera, Marc Rene Zofka, Tobias Strauss, Johann Marius Zöllner, and Christoph Stiller. "Making Bertha Cooperate-Team AnnieWAY's Entry to the 2016 Grand Cooperative Driving Challenge". In: *IEEE Transactions on Intelligent Transportation Systems* 19.4 (2018), pp. 1262–1276. DOI: 10.1109/TITS.2017.2749974.

Conference Publications

[3] Niels Ole Salscheider. *Object Tracking by Detection with Visual and Motion Cues*. 2021. arXiv: 2101.07549 [cs.CV]. URL: https://arxiv.org/abs/2101.07549 (visited on 01/20/2021).

[4] Niels Ole Salscheider. "FeatureNMS: Non-Maximum Suppression by Learning Feature Embeddings". In: *25th International Conference on Pattern Recognition, ICPR 2020*. Virtual Event / Milan, Italy: IEEE, 2020, pp. 7848–7854. DOI: 10.1109/ICPR48806.2021.9412930.

[5] Niels Ole Salscheider. "Simultaneous Object Detection and Semantic Segmentation". In: *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM) 2020*. Ed. by Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred. Valletta, Malta: SciTePress, 2020, pp. 555–561. DOI: 10.5220/0009142905550561.

[6] Niels Ole Salscheider and Martin Lauer. “Instance Segmentation by Learning Pixel Neighbour Relations with a CNN”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Ed. by Wei-Bin Zhang, Alexandre M. Bayen, Javier J. Sánchez Medina, and Matthew J. Barth. Maui, Hi, USA: IEEE, 2018, pp. 3463–3468. DOI: 10.1109/ITSC.2018.8569291.

[7] Niels Ole Salscheider, Eike Rehder, and Martin Lauer. “Analysis of Regionlets for Pedestrian Detection”. In: *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods (ICPRAM) 2017*. Ed. by Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred. Porto, Portugal: SciTePress, 2017, pp. 26–32. DOI: 10.5220/0006094100260032.

[8] Hendrik Königshof, Niels Ole Salscheider, and Christoph Stiller. “Real-time 3D Object Detection for Automated Driving Using Stereo Vision and Semantic Information”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. Auckland, New Zealand: IEEE, 2019, pp. 1405–1410. DOI: 10.1109/ITSC.2019.8917330.

[9] Fabian Poggenhans, Niels Ole Salscheider, and Christoph Stiller. “Precise Localization in High-Definition Road Maps for Urban Regions”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain: IEEE, 2018, pp. 2167–2174. DOI: 10.1109/IROS.2018.8594414.

[10] Annika Meyer, Niels Ole Salscheider, Piotr Franciszek Orzechowski, and Christoph Stiller. “Deep Semantic Lane Segmentation for Mapless Driving”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain: IEEE, 2018, pp. 869–875. DOI: 10.1109/IROS.2018.8594450.

[11] Ayman Tarakji, Niels Ole Salscheider, Stephan Alt, and Jan Heiducoff. “Feature-based device selection in heterogeneous computing systems”. In: *Computing Frontiers Conference (CF’14)*. Ed. by Pedro Trancoso, Diana Franklin, and Sally A. McKee. Cagliari, Italy: ACM, 2014, 9:1–9:10. DOI: 10.1145/2597917.2597927.

[12] Ayman Tarakji and Niels Ole Salscheider. “Runtime Behavior Comparison of Modern Accelerators and Coprocessors”. In: *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*.

Phoenix, AZ, USA: IEEE Computer Society, 2014, pp. 97–108. DOI: 10.1109/IPDPSW.2014.16.

[13] Ayman Tarakji, Niels Ole Salscheider, and David Hebbeker. “OS Support for Load Scheduling on Accelerator-based Heterogeneous Systems”. In: *Proceedings of the International Conference on Computational Science (ICCS) 2014*. Ed. by David Abramson, Michael Lees, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot. Cairns, Queensland, Australia: Elsevier, 2014, pp. 231–245. DOI: 10.1016/j.procs.2014.05.021.

[14] Alexander Naumann, Laura Dörr, Niels Ole Salscheider, and Kai Furmans. *Refined Plane Segmentation for Cuboid-Shaped Objects by Leveraging Edge Detection*. 2020. arXiv: 2003.12870 [cs.CV]. URL: https://arxiv.org/abs/2003.12870 (visited on 12/31/2020).

[15] Carlos Fernandez Lopez, Carlos Guindel, Niels Ole Salscheider, and Christoph Stiller. “A Deep Analysis of the Existing Datasets for Traffic Light State Recognition”. In: *21st International Conference on Intelligent Transportation Systems (ITSC) 2018*. Ed. by Wei-Bin Zhang, Alexandre M. Bayen, Javier J. Sánchez Medina, and Matthew J. Barth. Maui, HI, USA: IEEE, 2018, pp. 248–254. DOI: 10.1109/ITSC.2018.8569914.

[16] Eike Rehder, Maximilian Naumann, Niels Ole Salscheider, and Christoph Stiller. *Cooperative Motion Planning for Non-Holonomic Agents with Value Iteration Networks*. 2017. arXiv: 1709.05273 [cs.RO]. URL: http://arxiv.org/abs/1709.05273 (visited on 12/31/2020).

[17] Andre-Marcel Hellmund, Sascha Wirges, Ömer Sahin Tas, Claudio Bandera, and Niels Ole Salscheider. “Robot operating system: A modular software framework for automated driving”. In: *19th IEEE International Conference on Intelligent Transportation Systems (ITSC) 2016*. Rio de Janeiro, Brazil: IEEE, 2016, pp. 1564–1570. DOI: 10.1109/ITSC.2016.7795766.

Supervised Theses

[18] Tobias Fuchs. “Konstruktionstechnische Analyse, Weiterentwicklung und Validierung einer Pneumatischen Ausschleuseinheit zur automa-

tischen Sortierung heterogener Materialströme". Bachelor's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2020.

[19] Katharina Bachmann. "Unsupervised Semantic Segmentation of Point Cloud Data". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2019.

[20] Stefan Göbel. "Trainingsdatengenerierung auf Basis von Generative Adversarial Networks". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2019.

[21] Andreas Blattmann. "Multi-Person Pose Estimation using synthetically generated Data". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2019.

[22] Stanislav Frolov. "Deep Recurrent Neural Networks for Multiple Object Tracking". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2017.

[23] Timo Rilinger. "End-to-End Learning von Fahrzeugtrajektorien aus Kamerabildern mittels Convolutional Neural Networks". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2017.

[24] Jonathan Pydd. "Instanzweise Segmentierung in Verkehrsumgebungen mittels Deep Learning". Master's thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2017.

[25] Julio Sanchez. "Pixelweises Labeling und Tiefenwahrnehmung aus Stereobildpaaren mit Konvolutionsnetzen". Diploma thesis. Karlsruhe, Germany: Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), 2017.