Hochschule Düsseldorf
Faculty
Electrical Engineering and Information Technology

PERFORMANCE EVALUATION OF STATE-OF-THE-ART COMPUTER VISION SYSTEMS IN THE FIELD OF AUTONOMOUS DRIVING IN DEPENDENCY OF OPTICAL PARAMETERS

Thesis
Electrical Engineering and Information Technology
(M.Sc.)

Mattis Brummel
Student ID no.: [redacted]

Prof. Dr. rer. nat. Alexander Braun
Patrick Müller, M. Sc.

August 23, 2021

Mattis Brummel: *Performance evaluation of state-of-the-art Computer Vision Systems in the field of Autonomous Driving in dependency of Optical Parameters*, – A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Information Technology at the Düsseldorf University of Applied Science, Department of Electrical Engineering and Information Technology, August 2021

SUPERVISORS:
Prof. Dr. rer. nat. Alexander Braun
Patrick Müller, M. Sc.

LOCATION:
Düsseldorf

SUBMISSION:
August 2021

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources are acknowledged as references.

Düsseldorf, August 23, 2021

_____

Mattis Brummel

Abstract

With the development in deep learning, Computer Vision systems have received huge performance growth. The increasing research interest in perception systems for Autonomous Driving largely contributes to this progress. However, although it is essential for Computer Vision systems to reliably perform in safety-critical applications such as Autonomous Driving, little research has been done on linking the performance of these systems to the performance of optical systems and thus to the quality of images. In this thesis, the performances of Computer Vision systems are evaluated under simulated, physically realistic, effects of defocus. To this end, large-scale Autonomous Driving datasets are degraded by an optical model to simulate driving scenes under different effects of defocus, and the performances of Computer Vision systems on these degraded datasets are compared with the optical performance of the applied optical model. A new evaluation metric, called Spatial Recall Index (SRI), is proposed to evaluate the performance of object detection and instance segmentation systems in dependency of spatial positions in the input images. Using Hybrid Task Cascade (HTC) and Cascade Mask R-CNN for object detection and instance segmentation concerning pedestrians and cars in driving scenes, a dependency of the performances on the image quality was found both with standard evaluation metrics and the new SRI. Moreover, with the SRI metric a correlation could be observed between the spatially varying optical performance of the optical model and the spatial performance of Computer Vision systems. This highlights the importance of evaluating the robustness of Computer Vision systems to naturally occurring effects of defocus by also taking into account the spatial domain.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CNN	Convolutional Neural Network
DNN	Deep Neural Network
R-CNN	Region-based Convolutional Neural Network
HTC	Hybrid Task Cascade
AI	Artificial Intelligence
ReLU	Rectified Linear Units
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
RoI	Region of Interest
SVM	Support Vector Machine
NMS	non-maximum suppression
RPN	Region Proposal Network
IoU	Intersection over Union
FCN	Fully Connected Network
PSF	point spread function
AP	Average Precision
MAP	Mean Average Precision
MR	Miss Rate
FPPI	False Positives per Image
LAMR	Log Average Miss Rate
TP	True Positive
FP	False Positive
FN	False Negative
P	Positive
SRI	Spatial Recall Index
FOV	Field of view
FWHM	Full Width Half Maximum

Notation

$a_j^{(l)}$	Activation of neuron j in layer l
$z_j^{(l)}$	Weighted sum of the input to neuron j in layer l before applying the activation function h
h	Non-linear activation function such as ReLU
$w_{jk}^{(l)}$	Weight of the connection between the j-th neuron's activation in the current layer l and the k-th neuron's activation in the previous layer $l-1$
$b_j^{(l)}$	Bias of neuron j in layer l
$\boldsymbol{a}^{(l)}$	Vector of all neuron's activations in layer l
$\boldsymbol{a}^{(L)}$	Vector of all neuron's activations in the output layer L
$\boldsymbol{z}^{(l)}$	Vector representing the weighted sum of the inputs to all neurons in layer l before applying the element-wise activation function $\boldsymbol{h}$
$\boldsymbol{h}$	Element-wise activation function applied on the weighed sum of the inputs $\boldsymbol{z}^{(l)}$ to the neurons in layer l
$\boldsymbol{W}^{(l)}$	Weight matrix whose components $w_{jk}^{(l)}$ represent the weights of the connections between the j-th neuron's activations in the current layer l and the k-th neuron's activations in the previous layer $l-1$
$\boldsymbol{b}^{(l)}$	Vector whose components are the neuron's individual biases $b_j^{(l)}$ in layer l
$\mathcal{L}$	Loss (or cost) such as the negative cross-entropy or MSE over all n individual trainings examples x representing the networks overall agreement with the training data
$\mathcal{L}_x$	Loss (or cost) for one individual trainings example x
$\boldsymbol{a}^{(L)}(\boldsymbol{x})$	Output activation of a neural network with L layers after feeding and forward propagating input vector $\boldsymbol{x}$
$\boldsymbol{y}(\boldsymbol{x})$	Target vector (ground truth) associated to input vector $\boldsymbol{x}$
$\|\boldsymbol{v}\|^2$	Squared L_2 norm of vector $\boldsymbol{v}$

$\frac{\partial \mathcal{L}_x}{\partial z_j^{(l)}}$	Partial derivative of the loss $\mathcal{L}_x$ for one training example x with respect to the summed inputs $z_j^{(l)}$ to the neuron j in layer l
$\frac{\partial \mathcal{L}_x}{\partial a_j^{(l)}}$	Partial derivative of the loss $\mathcal{L}_x$ for one training example x with respect to the j-th neurons activation $a_j^{(l)}$ in layer l
$\frac{\partial \mathcal{L}_x}{\partial w_{jk}^{(l)}}$	Partial derivative of the loss $\mathcal{L}_x$ for one training example x with respect to the weight $w_{jk}^{(l)}$ representing the connection between the k-th neuron in the previous layer and the j-th neuron in the current layer
$\frac{\partial \mathcal{L}_x}{\partial b_j^{(l)}}$	Partial derivative of the loss $\mathcal{L}_x$ for one training example x with respect to the bias of neuron j in layer l
$\delta_j^{(l)}$	Error of neuron j in layer l
$\delta_j^{(L)}$	Error of neuron j in the output layer L
$\boldsymbol{\delta}^{(l)}$	Vector whose components are the errors of neurons in layer l
$\boldsymbol{\delta}^{(L)}$	Vector whose components are the errors of neurons in the output layer L
$\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x$	Matrix whose components are the partial derivatives of the loss $\mathcal{L}_x$ with respect to the weights in matrix $\boldsymbol{W}^{(l)}$ connecting the neurons in layer $l-1$ with those in layer l
$\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x$	Vector whose components are the partial derivatives of the loss $\mathcal{L}_x$ with respect to the biases of neurons in layer l
η	Learning rate
$w_{m,n}^{(l)}$	Learnable parameters in a kernel of a convolutional layer l
$a_{x,y}^{(l)}$	Output activation of a convolutional layer l at position x, y of the feature map
F	Spatial extend of a kernel in a convolutional or pooling layer
S	Stride of a kernel in a convolutional or pooling layer
P	Amount of padding applied on the input of a convolutional layer
D	Depth of a feature map
$\mathcal{L}_{cls}$	Classification loss
$\mathcal{L}_{loc}$	Localization loss or bounding box regression loss

$\mathcal{L}_{mask}$	Mask loss
BB_{gt}	Ground truth bounding box
BB_{dt}	Detected bounding box
BM_{gt}	Ground truth binary mask
BM_{dt}	Detected binary mask
$\mathbf{BB}_{gt}$	Array of size $g \times 4$ representing all g ground truth bounding boxes of size 1×4 for one category in one image
$\mathbf{BB}_{dt}$	Array of size $d \times 4$ representing all d detection bounding boxes of size 1×4 for one category in one image
$\mathbf{BM}_{gt}$	Array of size $g \times h \times w$ representing all g ground truth binary masks of size $h \times w$ for one category in one image
$\mathbf{BM}_{dt}$	Array of size $d \times h \times w$ representing all d truth binary masks of size $h \times w$ outputted by the instance segmentation system for one category in one image
Z_2^0	Zernike polynomial for defocus
Z_Δ	Defocus offset for the parameterizarion of the optical model

1 Introduction

In recent decades, huge progress has been made in the field of Autonomous Driving. A crucial factor for this progress is the development of Computer Vision. Tesla, for instance, one of the leading companies in the field of Autonomous Driving, relies exclusively on vision in their perception systems for semi-autonomous cars. However, given how safety-critical automotive systems are and how much they rely on the robustness of perception systems to navigate through real-world environments, surprisingly little research has been done in evaluating the performance of Computer Vision systems in dependency on the image quality of their inputs, despite the fact that potential mass productions of cameras for autonomous vehicles, due to time and cost constraints, almost unavoidably lead production tolerances in lenses. The optics of camera systems are always spatially variable over the field of view, so the influence of lenses on the performance of Computer Vision systems may even vary for different spatial positions.

Motivated by this, the goal of this thesis is to evaluate the dependency of Computer Vision systems in the field of Autonomous Driving on the spatially varying optical performance. To this end, a newly proposed evaluation metric, called Spatial Recall Index, is presented, which evaluates the object detection and instance segmentation performance in dependency on spatial positions in input images. Moreover, with an optical lens model, large-scale Autonomous Driving datasets are degraded to simulate driving scenes under physically realistic and spatially varying effects of defocus. Two selected state-of-the-art Computer Vision systems for object detection and instance segmentation are evaluated on these degraded datasets to compare their performance with the optical performance of the underlying lens model. Using HTC and Cascade Mask R-CNN, the evaluation focuses on object detection and instance segmentation and on pedestrians and cars in driving scenes [1, 2]. The evaluation is carried out with standard metrics such as the Precision vs. Recall and the MR vs. FPPI curves as well as the newly proposed SRI metric to assess the overall performance and the spatial performance of the Computer Vision systems under effects of defocus, respectively.

The work is organized as follows. After a thorough introduction into the fields of Computer Vision and Optics in section 2, the SRI metric is described in section 3. In section 4, the evaluation process from the selection of datasets and Computer Vision algorithms, to the image degradation, and up to the actual experimental setup is described. In section 5, the results of the experiments are presented. Finally, after a discussion of these results in section 6.1, a conclusion is given in section 6.2, before section 7 identifies possible future work.

2 Computer Vision and Optics

From a scientific perspective beyond pure engineering, Computer Vision and Optics are closely tied. Optics, on the one hand, refers to *forward* models that model, simply put, how light is refracted through camera lenses and projected onto an image plane respectively the sensor in order to create a scene. Computer Vision, on the other hand, tries to do the *inverse* by describing the scene in an image and reconstructing its properties. While humans do this effortlessly up to a full scene understanding, it is surprisingly tough to even achieve an understanding of subtasks such as scene recognition, image classification, object detection, and instance-level semantic labeling (or instance segmentation) on the way to a fully semantic scene understanding. This becomes even more difficult when the images created in physics and used as inputs for Computer Vision algorithms are not focused. In fact, the aforementioned lenses in cameras through which light passes before reaching the sensor are obviously not ideal (or infinitely thin). Instead, real lenses suffer from aberration such as spherical aberration, coma, astigmatism, curvature of field, and distortion. These naturally occurring lens aberrations along with other effects of defocus are camera specific phenomena leading to blurred regions in images that, in turn, Computer Vision algorithms have to deal with [3].

This section links Computer Vision and Optics by describing, in the former field, vision tasks for autonomous driving, and in the latter field, realistic defocus simulations that in turn can be used to evaluate Computer Vision algorithm's robustness to those simulated defocus conditions. The goal is to first, in section 2.1, provide an understanding of Computer Vision algorithms leading to modern frameworks respectively architectures that are used in this thesis. Since over the last decade Deep Neural Networks (DNNs) have become the method of choice for most Computer Vision tasks, section 2.1.1 starts by introducing deep learning with an explanation of deep feedforward networks generally before section 2.1.2 proceeds with the specific kind of feed forward network called Convolutional Neural Network (CNN) due to its prevalence in modern Computer Vision architectures for recognition tasks [3, 4]. Then, section 2.1.3 and 2.1.4 present how these deep learning techniques can be applied to Object Detection and Instance Segmentation, respectively, two tasks with particular importance for self driving cars. The former, Object Detection, refer to instance localization and classification by drawing bounding boxes around recognized objects, whereas the latter, Instance Segmentation, extends the task in terms of accuracy by assigning each pixel to a semantic label of the instance [5]. Finally, section 2.1.5 presents modern Computer Vision architectures from which two are used for evaluation

in this work: Cascade Mask R-CNN [2, 6] and HTC [1].

The success in various Computer Vision tasks in the past decade is mainly driven by the availability of large-scale datasets [5]. In fact, modern deep learning provides a powerful framework for supervised learning, but it relies heavily on high-quality annotated data [4]. Section 2.2 provides an overview of high-quality datasets for autonomous driving before section 2.3 explains common evaluation methodologies to assess the performance of Computer Vision algorithms on these given datasets. Here, the focus is on Object Detection and Instance Segmentation.

Finally, section 2.4 makes the connection from Computer Vision to Optics. It describes the realistic defocus simulation of camera objectives, which can be used to assess the performance of Computer Vision systems under the effects of defocus.

2.1 Computer Vision

Computer Vision is a field of Artificial Intelligence (AI) that focuses on vision tasks, aiming to automatically extract meaningful information in images. Deriving these informations requires high computational demands and sophisticated methods for efficiency, as a single RGB image usually contains several million values that need to be processed [5]. DNNs and especially deep networks with convolutional layers have proved most effective in the majority of Computer Vision tasks and are thus the most widely used machine learning models in the field [3, 4]. They are described along with important deep learning techniques in section 2.1.1 and 2.1.2, before section 2.1.3 and 2.1.4 narrow the field by explicitly focussing on Object Detection and Instance Segmentation, respectively. Finally, modern architectures with state-of-the-art performance in these tasks are presented in section 2.1.5.

2.1.1 Deep Learning

Deep learning refers to techniques that enable learning in DNNs [7]. Inspired by neuroscience, DNNs are by definition neural networks with two or more hidden layers between input and output layer [7, 4, 8]. State-of-the-art DNNs are usually composed of thousands of interconnected "neurons" (or units) organized in many layers forming the aforementioned multiple-layer structure [4, 7]. The most popular DNNs are feedforward networks such as CNNs trained using gradient descent and backpropagation leading to breakthroughs in the field of Computer Vision in the past decade [3]. This section exemplifies deep learning concepts in a simple feedforward neural network, before section 2.1.2 discusses CNNs in order to emphasize how convolutional layers boost the efficiency of neural networks in image driven systems in deployment as well as in the training step. Starting with an explanation of deep learning concepts in a fully-connected feedforward

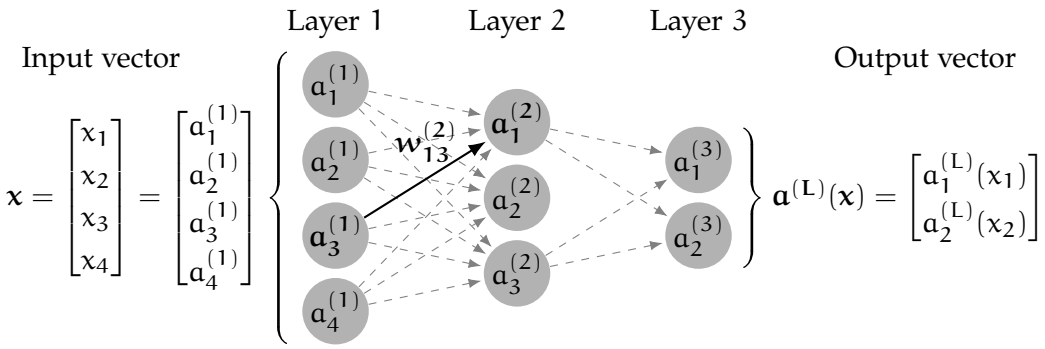


Figure 2.1: An example of a fully-connected feedforward network with one hidden layer. The term a denotes the neuron's activation and w the weights. The input layer's activations represent the neural network's input vector $\mathbf{x}$, whereas the remaining activations describe the results from the respective activation functions. Note that each neuron in layers 2 and 3 contains an individual bias b.

network before introducing convolutional layers makes the improvements of CNNs compared to Fully Connected Networks (FCNs) in terms of efficiency and thus computational demand clear.

2.1.1.1 Forward Pass

In order to describe the forward pass in a neural network, figure 2.1 shows a simple fully-connected feedforward network, which for the sake of simplicity contains only one hidden layer with three neurons, an input layer with four neurons, and an output layer with two neurons. In each fully-connected layer, in which by definition all input units are connected to all output units, the j-th neuron's activation $a_j^{(l)}$ in layer l is computed by the weighted sum of their input activations $a_k^{(l-1)}$ and a bias $b_j^{(l)}$ followed by a non linear activation function h [3, 7],

$$a_j^{(l)} = h\left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \right). \tag{1}$$

As highlighted in figure 2.1, the weights $w_{jk}^{(l)}$ in equation 1 represent the connection between the j-th neuron's activation in the current layer and the k-th neuron's activation in the previous layer. The order of the chosen indices j and k for the weights w seem somewhat counterintuitive, but they help to rewrite equation 1 in matrix from by defining the weight matrix $\boldsymbol{W}^{(l)}$, denoting all connections between the fully-connected layers $l-1$ and l (see appendix A). Each neuron's individual bias b in layer l is summarized by the vector $\mathbf{b}^{(l)}$ and the weighted sum of the inputs to the neurons in layer l before applying the activation function $\mathbf{h}$ is called $\boldsymbol{z}^{(l)}$. A forward pass through one layer of a FCN is

Algorithm 2.1 `forwardPass` (Forward Pass) [7, 4]

Input: $\boldsymbol{W}^{(l)}, \mathbf{b}^{(l)}, \quad l \in \{2, 3, ..., L\}$ ▷ weight matrices and bias vectors
Input: $\boldsymbol{x}$ ▷ input vector

function FORWARDPASS($\boldsymbol{W}^{(l)}$, $\mathbf{b}^{(l)}$, $l \in \{2, 3, ..., L\}$, $\boldsymbol{x}$)
  $\mathbf{a}^{(1)} \leftarrow \boldsymbol{x}$ ▷ input $\boldsymbol{x}$ representing the first layer's activations
  for $l = 2, ..., L$ **do** ▷ forward pass through layers in the network
    $\boldsymbol{z}^{(l)} \leftarrow \boldsymbol{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ ▷ weighted sum $z^{(l)}$ of inputs to layer l (eq. 2)
    $\mathbf{a}^{(l)} \leftarrow \mathbf{h}\big(\boldsymbol{z}^{(l)}\big)$ ▷ element-wise activation $\mathbf{h}$ on vector $\boldsymbol{z}^{(l)}$ (eq. 3)
  end for ▷ store vectors $\mathbf{a}^{(l)}$ and $\boldsymbol{z}^{(l)}$ in each iteration
  return $(\boldsymbol{z}^{(l)}(\boldsymbol{x}), \mathbf{a}^{(l)}(\boldsymbol{x}))$ ▷ return $\boldsymbol{z}^{(l)}$ and $\mathbf{a}^{(l)}$ of all $L-1$ layers for input $\boldsymbol{x}$
end function

now defined as

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{2}$$

$$\mathbf{a}^{(l)} = \mathbf{h}\big(\boldsymbol{z}^{(l)}\big), \tag{3}$$

where the activation function $\mathbf{h}$ is typically a non-linear function such as Rectified Linear Units (ReLU) and applied element-wise [9, 4]. ReLU is defined as

$$h(z) = max\{0, z\}, \tag{4}$$

and thus basically a piecewise linear function with two linear pieces for input values greater than and less than or equal to 0 [9, 4]. In fact, the activation function along with the number of neurons are basically the only attributes that distinguish different layers of a FCN [4]. For the sake of clarity, there is only one hidden layer in the example network in figure 2.1 and the dimensions of the weight matrices denoting their connections to the input and output layer are 3×4 and 2×3, respectively (i.e. it is not a DNN by the aforementioned definition).

After initializing the weights and biases, which apart from the network's architecture itself (i.e. the number of layers and neurons, and the type of connections) continues to be an active research area, a complete forward pass is computed by feeding an input vector $\boldsymbol{x}$ (see input vector in figure 2.1) and forward propagating it through each layer in the network. Therefore, applying the equations 2 and 3 repeatedly from the first layer onwards results in the output (or prediction) $\mathbf{a}^{(L)}(\boldsymbol{x})$ with L denoting the number of layers in the network (i.e. the last layer) and $\mathbf{a}^{(L)}(\boldsymbol{x})$ representing the vector of all output activations for input vector $\boldsymbol{x}$ (see output vector in figure 2.1) [3, 4]. Algorithm 2.1 shows how to compute the output of a neural network. The function returns not only the output vector $\mathbf{a}^{(L)}$ but also the vector $\boldsymbol{z}^{(L)}$ as well as the vectors $\boldsymbol{z}^{(l)}$ and $\mathbf{a}^{(l)}$ from all hidden layers, because they are used during the training as explained below.

2.1.1.2 Loss Function

In supervised learning, which is the term for learning strategies that are based on pre-labeled input data such as Stochastic Gradient Descent (SGD), each of the inputs x is paired with an output (or target) $y(x)$. Suppose, for instance, the neural network learns to classify input images into two categories, say, "cat" and "dog", then the input is an image converted into a pixel vector x and the associated target is a two dimensional vector $y(x)$ with a 1 at the index of the true category and a 0 for the wrong category in the image. The output of the network, after feeding and forward propagating x through the network based on algorithm 2.1, is also a two dimensional vector, and the training algorithm maximizes the agreement between the network's output and the target by adjusting the initialized parameters (i.e. the learnable weights and biases) [3]. This leads to the the so-called *loss function* (or cost), which quantifies how well the network's output $a^{(L)}(x)$ approximate the associated target $y(x)$ [3, 7]. The loss function of most modern neural networks is the negative cross-entropy between targets and outputs [4]. In order to simplify the math for subsequent explanations, however, equation 5 introduces a loss function called Mean Squared Error (MSE)

$$\mathcal{L} = \frac{1}{2n} \sum_{x} \left\| y(x) - a^{(L)}(x) \right\|^2 , \tag{5}$$

where $y(x)$ represents the target vector associated to the output vector $a^{(L)}$. MSE often leads to poor results in gradient descent algorithms and is thus usually not an option in practical application [4]. However, it has an intuitive interpretation that holds true for the negative cross-entropy as well [7]. In short, the loss for one training example (i.e. input) x tends towards 0 when the output of the network $a^{(L)}(x)$ (i.e. the vector of all values from the output layer's units), after feeding and forward-propagating the input through all layers of the network, approximates the associated target $y(x)$. In equation 5, the loss for one individual training input is denoted by the term $\frac{1}{2}\|y(x) - a^{(L)}(x)\|^2$ where the notation $\|v\|^2$ refers to the squared L_2 norm of vector v [7, 3]. The term for the loss of one training example may thus be rewritten as $\frac{1}{2}\sum_j (y_j(x) - a_j^{(L)}(x))^2$ by using the indices j for the respective units in the output layer as well as for the associated components in target vector $y(x)$. Instead of computing the loss for only one input, however, the MSE in equation 5 computes, just like other loss functions such as the negative cross-entropy, the average loss over all n individual training examples x, resulting overall in a measure of the network's agreement with the training data, where n refers to the total number of examples in a training dataset. And the aforementioned SGD represents a training algorithm that repeatedly adjusts the weights and biases of the network in order to decrease the loss with the goal of finding a minimum in equation 5 [7]. Before explaining the SGD, however,

the next section explains backpropagation, which is essentially a procedure to gain an understanding about how changing the weights and biases changes the loss appropriately [7].

2.1.1.3 Backpropagation

The loss is a function of the output layer's activations (see equation 5), which in turn are functions their summed inputs (see equation 3) as well as the output unit's biases $b_k^{(l)}$ and the weights $w_{jk}^{(l)}$ connecting the two last adjacent layers (see equation 2). Therefore, after each forward-pass (computed with equations 2 and 3 in algorithm 2.1) the partial derivative of the loss for one training example with respect to the weights $\partial \mathcal{L}_x / \partial w_{jk}^{(l)}$ and biases $\partial \mathcal{L}_x / \partial b_j^{(l)}$ can be computed using the chain rule

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}_x}{\partial z_j^{(l)}} = \frac{\partial \mathcal{L}_x}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial \mathcal{L}_x}{\partial a_j^{(l)}} \cdot h'\big(z^{(l)}\big)\,, \tag{6}$$

$$\frac{\partial \mathcal{L}_x}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}\,, \tag{7}$$

$$\frac{\partial \mathcal{L}_x}{\partial b_j^{(l)}} = \delta_j^{(l)}\,, \tag{8}$$

where $\delta_j^{(l)} = \partial \mathcal{L}_x / \partial z_j^{(l)}$ refers to the partial derivative of the loss $\mathcal{L}_x$ for one training example x with respect to the summed inputs $z_j^{(l)}$ to the neuron j in layer l [3, 7]. The intermediate quantity $\delta_j^{(l)}$ computed by applying the chain rule on the partial derivative of the loss with respect to the j-th neurons activation $\partial \mathcal{L}_x / \partial a_j^{(l)}$ and the partial derivative of this activation with respect to its summed inputs $\partial a_j^{(l)} / \partial z_j^{(l)}$ is the so-called *error* [7]. After a complete forward pass, equation 6 can be used to compute the errors of the output layer, which is indicated by $l = L$ to be consistent with equation 5. Equation 7 and 8 may then be used to compute the partial derivatives of the loss for one training example with respect to the weights connecting the last two adjacent layer and the biases in the last layer. In order to compute the derivatives of the loss with respect to weights and biases in previous layers, the errors of units in previous layers must first be computed. In fact, the term $\delta_j^{(l)}$ is referred to as the error in the j-th neuron in the l-th layer, as it gets propagated backward through the network [7, 3].

In short, computing first the partial derivative of the loss with respect to the error of units in the output layer (with equation 6) and plugging them into equation 7 and 8 leads to the partial derivatives of the loss with respect to the weights connecting the last two adjacent layers as well as the biases of the units in the last layer. And propagating the error backward simply means working

the way back through the network in order to repeatedly compute the errors of neurons in the previous layer before plugging these errors into equations 7 and 8, respectively, until the partial derivatives of the loss with respect to all weights and biases of the network are computed. In a fully-connected network, the k-th neuron in the previous layer feeds all neurons in the current layer, so the k-th neuron's error is a function of all errors in the current layer. More specifically, the k-th neuron's error in the current layer l is the weighted sum of all errors in the next layer $l+1$ multiplied by the derivative of its activation function $h'(z_k^{(l)})$ [3].

$$\delta_k^{(l)} = \left(\sum_j w_{jk}^{(l+1)} \delta_j^{(l+1)} \right) \cdot h'(z_k^{(l)}) \tag{9}$$

In summary, equation 6 describes the error of the j-th unit in the output layer and equation 9 explains how to propagate the error backward in order to compute the error of the k-th unit in previous layers. Finally, equation 7 and 8 represent the partial derivative of the loss (for one training example) with respect to the weights and biases, respectively. Just like the equations 2 and 3 for computing the output of a FCN, the four equations behind back-propagation can be written in matrix form:

$$\boldsymbol{\delta}^{(L)} = \nabla_a \mathcal{L}_x \odot \mathbf{h}(\boldsymbol{z}^{(L)}) \tag{10}$$

$$\boldsymbol{\delta}^{(l)} = \left((\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \right) \odot \mathbf{h}'(\boldsymbol{z}^{(l)}) \tag{11}$$

$$\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x = \boldsymbol{\delta}^{(l)} \tag{12}$$

$$\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x = \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T \tag{13}$$

The vector $\boldsymbol{\delta}^{(L)}$ denotes the errors in the output layer and $\boldsymbol{\delta}^{(l)}$ describes the vector of errors in layer l as a function of the vector of errors in the adjacent layer $l+1$. The term $\nabla_a \mathcal{L}_x$ denotes a vector whose components are the partial derivatives $\partial \mathcal{L}_x / \partial a_j^{(l)}$ of the loss $\mathcal{L}_x$ for training example x with respect to the output layer's activations $a^{(L)}$ introduced in equation 3 [7]. The term $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x$ represents a matrix whose components are the partial derivatives of the loss $\mathcal{L}_x$ with respect to the weights in matrix $\boldsymbol{W}^{(l)}$ connecting the neurons in layer $l-1$ with those in layer l. Finally, the term $\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x$ denotes the vector of partial derivatives of the loss $\mathcal{L}_x$ with respect to the biases of neurons in layer l.

As shown in algorithm 2.1, all intermediate values computed during the forward pass must be kept in memory as they are required for backpropagating the error. More specifically, equation 9 indicates that the weighted sums $z_k^{(l)}$ are required for backpropagating the error, and equation 7 suggests that the neuron's activations $a_k^{(l)}$ are needed for computing the partial derivative of the loss with respect to the weights. Storing all these values is in fact a price in memory

Algorithm 2.2 `backProp` (Backpropagation) [7, 4]

Input: $\boldsymbol{W}^{(l)}, \mathbf{b}^{(l)}, \quad l \in \{2, 3, ..., L\}$ ▷ weight matrices and bias vectors
Input: $\mathbf{a}^{(l)}(\mathbf{x}), \quad l \in \{2, 3, ..., L\}$ ▷ vectors $\mathbf{a}^{(l)}$ for $\mathbf{x}$ returned by algorithm 2.1
Input: $\mathbf{z}^{(l)}(\mathbf{x}), \quad l \in \{2, 3, ..., L\}$ ▷ vectors $\mathbf{z}^{(l)}$ for $\mathbf{x}$ returned by algorithm 2.1
Input: $\mathbf{y}(\mathbf{x})$ ▷ target vector associated to input vector $\mathbf{x}$

function BACKPROP($\boldsymbol{W}^{(l)}$, $\mathbf{b}^{(l)}$, $\mathbf{a}^{(l)}(\mathbf{x})$, $\mathbf{z}^{(l)}(\mathbf{x})$, $l \in \{2, 3, ..., L\}$, $\mathbf{y}(\mathbf{x})$)
  $\boldsymbol{\delta}^{(L)} \leftarrow \nabla_{\mathbf{a}} \mathcal{L}_x \odot \mathbf{h}\big(\mathbf{z}^{(L)}\big)$ ▷ vector with errors $\delta_j^{(L)}$ in output layer L (eq. 10)
  $\nabla_{\mathbf{b}^{(L)}} \mathcal{L}_x \leftarrow \boldsymbol{\delta}^{(L)}$ ▷ vector with elements $\partial \mathcal{L}_x / \partial b_j^{(L)}$ (eq. 12)
  $\nabla_{\boldsymbol{W}^{(L)}} \mathcal{L}_x \leftarrow \boldsymbol{\delta}^{(L)} \big(\mathbf{a}^{(l-1)}\big)^T$ ▷ matrix with elements $\partial \mathcal{L}_x / \partial w_{jk}^{(L)}$ (eq. 13)
  for $l = L-1, L-2, ..., 2$ **do** ▷ backpropagate the error
    $\boldsymbol{\delta}^{(l)} \leftarrow \big(\big(\boldsymbol{W}^{(l+1)}\big)^T \boldsymbol{\delta}^{(l+1)}\big) \odot \mathbf{h}'(\mathbf{z}^{(l)})$ ▷ error of layer l (eq. 11)
    $\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x \leftarrow \boldsymbol{\delta}^{(l)}$ ▷ vector with elements $\partial \mathcal{L}_x / \partial b_j^{(l)}$ (eq. 12)
    $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x \leftarrow \boldsymbol{\delta}^{(l)} \big(\mathbf{a}^{(l-1)}\big)^T$ ▷ matrix with elements $\partial \mathcal{L}_x / \partial w_{jk}^{(l)}$ (eq. 13)
  end for ▷ store vectors $\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x$ and matrices $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x$ in each iteration
  return $(\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x, \nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x)$ ▷ return $\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x$ and $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x$ for all layers
end function

capacity that is paid for the computational efficiency of the backpropagation algorithm [10].

Nevertheless, algorithm 2.2 summarizes this section by showing the backpropagation procedure in its entirety. First, the errors in the output layer as well as the partial derivatives of the loss for one training example with respect to the weights connecting the last two adjacent layer and the biases in the last layer are computed with equation 10, 12, and 13, respectively. Then, with each iteration in a for loop, the error is propagated one layer backwards with equation 11 in order to repeatedly apply equation 12 and 13 for computing the partial derivative of the loss for this training example with respect to the remaining weights and biases in the network. The next section describes the SGD, which is essentially a backpropagation training rule and according to Goodfellow et al. [4] the most used training algorithm in neural networks.

2.1.1.4 Stochastic Gradient Descent

So far, the network's architecture, the activation functions, and the loss function are defined. Moreover, the procedure of computing the derivative of the loss $\nabla_x \mathcal{L}$ for for one training example x with respect to all weights and biases of the network is described (i.e. $\partial \mathcal{L}_x / \partial w_{jk}^{(l)}, \partial \mathcal{L}_x / \partial b_j^{(l)}, l \in \{2, 3, ..., L\}$). The gradient represents, simply put, the direction of the loss and the way the SGD algorithm works is to compute it repeatedly with respect to all weights and biases and

update all learnable parameters in order to move with small steps in the opposite direction of the gradient [7]. In fact, the size of the step is represented by another parameter called learning rate η, which needs to be carefully adjusted to make good progress without risking to miss potential minima of the loss function [7]. However, deriving the direction of the loss for one training example is a noisy estimate and averaging the gradient over all training examples before updating the parameters is too time intensive, especially for large datasets [3]. This is where the term *Stochastic* in SGD comes in: In each training epoch, the dataset is first shuffled and then partitioned in so-called *minibatches* of size m. Finally, the SGD algorithm loops through all minibatches, averages in each iteration the loss and the gradient, and updates the weights and biases based on the average gradient multiplied by the learning rate η. Equation 14 and 15 define the respective update rules that update the current weight matrices and bias vectors indexed by t in order to move in the opposite direction of the gradient.

$$\boldsymbol{W}_{t+1}^{(l)} = \boldsymbol{W}_{t}^{(l)} - \frac{\eta}{m} \sum_{x}^{m} (\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}_x) \tag{14}$$

$$\mathbf{b}_{t+1}^{(l)} = \mathbf{b}_{t}^{(l)} - \frac{\eta}{m} \sum_{x}^{m} (\nabla_{\mathbf{b}^{(l)}} \mathcal{L}_x) \tag{15}$$

Indeed, more sophisticated optimization techniques have been developed over the last decade. One of the most popular ones is the gradient based approach Adam [11, 3]. Nevertheless, regardless of which optimization technique is applied, the whole process must be repeated for several epochs, from which each epoch loops though all minibatches, in order to reach the goal of finding a minimum of the respective loss function. For the sake of clarity, algorithm 2.3 summarizes the learning process with SGD in its entirety. It doesn't take into account, however, that training data is usually partitioned into a training and validation set. Section 2.2 focuses on datasets, so these details will be discussed there. It can be stated beforehand, though, that a split in training and validation data helps to prevent the model from overfitting by basing parameter updates only on the training data (with algorithm 2.3) and validating the model's progress with the loss function after each epoch on the validation data. Finally, after completing the entire training process, the overall performance is assessed on a test set of the dataset.

To conclude this section, a few final considerations regarding the performance of the SGD algorithm in modern deep learning frameworks are yet to be mentioned. For explanation purposes, the training algorithm described here loops through all training examples in a minibatch. In modern frameworks, however, rather than looping through each training examples, the minibatches are combined to tensors in order to compute them simultaneously [7]. This results in dramatic speed improvements in vector based languages such as Python,

Algorithm 2.3 SGD (Stochastic Gradient Descent) [7, 4]

Input: $\boldsymbol{W}^{(l)}, \mathbf{b}^{(l)}, \quad l \in \{2, 3, ..., L\}$ ▷ initialized weight matrices and bias vectors
Input: $\mathbf{x}_1, ..., \mathbf{x}_n$ ▷ input vectors of n training examples
Input: $\mathbf{y}(\mathbf{x}_1), ..., \mathbf{y}(\mathbf{x}_n)$ ▷ target vectors associated to input vectors $\mathbf{x}_1, ..., \mathbf{x}_n$
Input: η ▷ learning rate

function SGD ($\boldsymbol{W}^{(l)}$, $\mathbf{b}^{(l)}$, $l \in \{2, 3, ..., L\}$, $\mathbf{x}_1, ..., \mathbf{x}_n$, $\mathbf{y}(\mathbf{x}_1), ..., \mathbf{y}(\mathbf{x}_n)$, η)
  for $e = 1, ..., k_{epochs}$ **do** ▷ loop through epochs (shuffle data in each epoch)
    for $M_i = M_1, ..., M_{\frac{n}{m}}$ **do** ▷ loop through $\frac{n}{m}$ mini batches M of size m
      for $\mathbf{x}_t = \mathbf{x}_1, ..., \mathbf{x}_m$ **do** ▷ loop through m training examples of batch
        $\boldsymbol{z}^{(l)}_{\mathbf{x}_t}, \boldsymbol{a}^{(l)}_{\mathbf{x}_t} \leftarrow$ FORWARDPASS($\boldsymbol{W}^{(l)}, \mathbf{b}^{(l)}, \mathbf{x}_t$)
        $\mathcal{L}_{\mathbf{x}_t} \leftarrow \mathcal{L}\big(\mathbf{y}(\mathbf{x}_t), \boldsymbol{a}^{(L)}(\mathbf{x}_t)\big)$ ▷ loss for one training example $\mathbf{x}_t$
        $\nabla_{\mathbf{b}^{(l)}}\mathcal{L}_{\mathbf{x}_t}, \nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}_{\mathbf{x}_t} \leftarrow$ BACKPROP($\boldsymbol{W}^{(l)}, \mathbf{b}^{(l)}, \boldsymbol{a}^{(l)}_{\mathbf{x}_t}, \boldsymbol{z}^{(l)}_{\mathbf{x}_t}, \mathbf{y}_{\mathbf{x}_t}$)
      end for ▷ add up $\nabla_{\mathbf{b}^{(l)}}\mathcal{L}_{\mathbf{x}_t}$, $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}_{\mathbf{x}_t}$ and $\mathcal{L}_{\mathbf{x}_t}$ in all m iterations
      $\mathcal{L} \leftarrow \frac{1}{m}\sum_t^m(\mathcal{L}_{\mathbf{x}_t})$ ▷ loss for minibatch M_i
      $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \frac{\eta}{m}\sum_t^m(\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}_{\mathbf{x}_t})$ ▷ update weights by update rule
      $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \frac{\eta}{m}\sum_t^m(\nabla_{\mathbf{b}^{(l)}}\mathcal{L}_{\mathbf{x}_t})$ ▷ update biases by update rule
    end for ▷ updated parameter in the opposite direction of the gradient
  end for ▷ after running through all batches, proceed with next epoch
end function

especially with the availability of GPU-based parallel computing. Another consideration concerns the intermediate values that according to algorithm 2.3 are stored in the forward pass in order to use them for computing the errors and gradients during backpropagation. Since modern neural networks usually have millions of neurons and thus activations, the number of values to be stored may need to be reduced. This can be done with *gradient checkpointing*, where activations are only stored at certain layers while others are re-computed in the backpropagation phase [3].

Finally, with a certain kind of data such as RGB images, the network architecture itself must be changed or complemented by computational elements other than fully-connected layer discussed here. Considering the complexity of standard RGB images with several million pixel values, the training process with a fully-connected network becomes too ineffective. Even a small RGB image of size 64x64x3 connected to a hidden layer of the same size (i.e. with 64x64x3 units) already leads to 12,288 weights on each neuron of the first layer, which makes a total of ca. 150 million weights connecting the first two layers [8, 12]. Neurons in a convolutional layer, on the other hand, are only connected to a small region of neurons in the previous layer. This drastically reduces the overall number of parameters in the network. Nevertheless, the concepts outlined in this section including calculating the loss function, propagating the error backwards with the chain rule until the derivative of the loss with respect to all learnable parameters

is computed before finally adjusting the parameters accordingly, remain with convolutional layers essentially the same [3]. Along with other features that make CNNs effective, convolutional layers are introduced in section 2.1.2.

2.1.2 Convolutional Neural Networks

The convolutional layer uses convolution rather than matrix multiplication and a DNN is called CNN when at least one of its layers is a convolutional layer [4]. Whereas in a fully-connected networks, the input values, even when they represent pixels of an image, are usually summarized by a vector and connected to all neurons of the first hidden layer (cf. connection between input vector $\mathbf{x}$ and layer 2 in figure 2.1), neurons in a convolutional layer are only connected to a small region of neighboring neurons called the input's *receptive field*. This basically means that the former doesn't take into account the spatial structure of images, while the latter even takes advantage of it [7]. CNNs are therefore mostly used to perceive patterns in image driven systems [8].

Instead of computing the weighted sum of all activations in the previous layer, which is the operation performed by a fully-connected layer as shown in equation 2, in a convolutional layer, the weighted sum is only performed within a small local window [3]. The mathematical operation with which the sum is calculated is called discrete convolution, since the inputs to CNNs are multidimensional arrays usually referred to as tensors and thus consist of discrete values [4]. In the convolutional network terminology, the local window is called kernel of a certain size and slides across the input, while at each location the values of the kernel are multiplied with the input values they overlap [13]. The sum of these multiplications (often with an additional bias) results in the aforementioned weighted sum of the receptive field and is usually, just like the weighted sum in a fully-connected layer, passed through a non-linear function such as ReLU in order to produce the final output in the current location of the resulting layer [13]. To remain in the CNN terminology, the output after performing the weighted sum at each location is referred to as *output feature map* [4].

Along with the *spatial extend of the kernel* (i.e. the size of the receptive field), there are certain hyperparameter such as the *depth* of the output volume, the kernel (or filter) *stride* and *padding* that define how the convolutional layer operates, i.e. how the kernel slides across the input in order to compute the output layer's activations [12]. In fact, these parameters along with the size of the input layer even determine the shape of the feature map respectively the size of the output layer. This is another important difference to the fully-connected layers where the output layer of two adjacent layers is independent from the input layer [13]. Figure 2.2 shows an intuitive example of the convolution operation without padding, where a 3×3 kernel with a depth of 1 slides with a stride length of 1 over an input layer of size 5×5 with a depth of 1 (i.e. with one

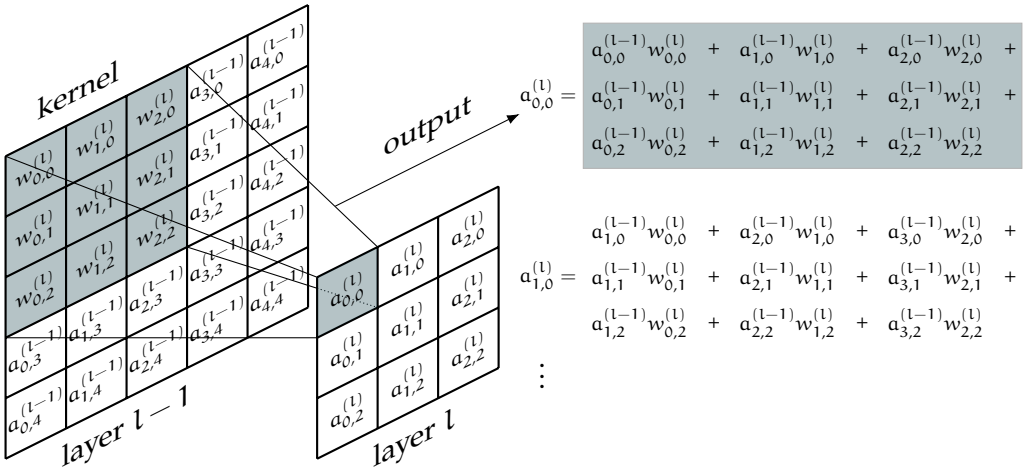


Figure 2.2: Visualization of the convolution operation in a CNN. A 3×3 kernel slides with a stride length of 1 over layer $l-1$ of size 5×5 with 0 padding and performs the weighted sum of the receptive fields by omitting the non-linearity h and a potential bias, which results in a layer l of size 3×3. An input to a convolutional layer requires a kernel with the same depth. In this example, the input has one channel (i.e. a depth of 1) and correspondingly the kernel have a depth of 1.

channel), resulting in a feature map of the volume $3 \times 3 \times 1$. It is an intuitive example of the convolution arithmetic that covers two of the four aforementioned hyperparameters (stride and spatial extend of the kernel), while making the necessity of padding in certain cases clear. The convolution operation with a depth larger than 1 is hard to visualize, so the last hyperparameter representing the depth of the output volume will be explained further below.

The 3×3 kernel, whose learnable weights $w^{(l)}_{m,n}$ are indexed by m and n to distinguish them from weights in a fully-connected layer, is highlighted at its current position on the top left corner of the input layer $l-1$. The respective weighted sum results in in the output activation $a^{(l)}_{0,0}$. For the sake of simplicity, the bias that is usually added as well as the non-linearity through which the weighted sum is generally passed, is omitted in this example. With a stride length of 1, the kernel slides one pixel to the right where its values overlap with the three central columns, resulting in output activation $a^{(l)}_{1,0}$. In the next step, the kernel overlaps the three right columns before sliding one pixel downwards and repeating the process with the remaining rows. Generally formulated in mathematical terms, the convolution operation for a two-dimensional image with a kernel of arbitrary size that slides with a stride length of 1 across the input in order to produce the weighted sums of of the local receptive fields, which in turn are added by a bias b and passed through a non-linearity h, results in the output

activations $a^{(l)}_{x,y}$

$$a^{(l)}_{x,y} = h\left(b + \sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w^{(l)}_{m,n} a^{(l-1)}_{x+m,y+n}\right), \tag{16}$$

where $F \times F$ represents the kernel size, $w^{(l)}_{m,n}$ denotes the learnable parameters in the kernel and $a^{(l-1)}_{x+m,y+n}$ are the input activations in the receptive field. To be precise, since the offsets m, n in equation 16 are added to and not subtracted from the pixel coordinates x, y, the operation is actually a cross-correlation rather than a convolution [3]. This distinction, however, is usually not mentioned and machine learning libraries perform cross-correlation while calling it convolution [3, 4]. Nevertheless, there are a few important things to note here with the mathematical expression in equation 16 and more intuitively with the visual example outlined in figure 2.2. They will be discussed in the following paragraphs.

Through the sliding window manner of the convolutional operation, the weights of the kernel (and the bias), with which the activations $a^{(l)}_{x,y}$ in layer l are computed, are actually the same across all neurons. This means that the number of parameters between convolutional layers compared to those between fully-connected layers is not only reduced by the sparse connections but also through *shared weights and biases* [7]. Apart from the reduced memory requirements for storing the parameters as well as the obvious speed improvements in the SGD algorithm due to the fact that simply less weights need to be updated, shared weights have yet another important effect [4]. The kernel operates at each location of the input with the exact same values and thus may extract the same features, only at different locations of the input [7]. In the first layer those features may be low-level features such as edges, while features in subsequent layers may find patterns in edges and more complex shapes [4]. This is in fact precisely why outputs of convolutional layers are referred to as feature maps [7].

Moreover, the example in figure 2.2 shows that the convolutional layer shrinks the resolution from a 5×5 input to a 3×3 output, and it can be inferred directly from this example that a decreased kernel of the size 2×2 would lead to a 4×4 feature map. It is therefore generally possible to use smaller kernels in order to let the spatial extent of the input shrink less rapidly with each convolution [4]. However, a method represented by the last one of the aforementioned hyperparameter may even preserve the size of the input. More specifically, the method is referred to as padding and it provides the possibility to control the kernel width and the size of the output independently [4]. Considering the 3×3 kernel with a stride length of 1 and the input size 5×5 in the example in figure 2.2, the spatial extent may be preserved by concatenating zeros at the beginning and at the end of the axis. This so-called zero padding would result in a 7×7 input map with zeros in the first and the last rows and columns, respectively,

on which the same kernel with the same stride length produces a 5×5 feature map. In fact, padding with respect to the input size, the kernel size and the stride length that preserve the input dimensionality in the aforementioned way is often referred to as half padding, whereas increasing the dimensionality (by concatenating more zeros at the beginning and at the end of the axis) is called full padding [13].

Decreasing the size of the input, on the other hand, is more straightforward. A stride length of 2, for instance, which means that the kernel moves two pixels to the right (or down), would lead in the example above (with 0 padding) to a 2×2 feature map. However, it is important to note that the spatial arrangement hyperparameters have mutual constraints [12]. Simply put, while a stride length of 2 is possible in this example, it may not be feasible on an input with a different size. This is the case because, as mentioned before, the hyperparameters define the dimension of the output and the output's neurons with their receptive fields must fit neatly across the entire input [12]. More specifically, the output of a convolutional layer is determined by

$$W_2 = \frac{W_1 - F + 2P}{S} + 1\,, \tag{17}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1\,, \tag{18}$$

where F is the spatial extent of the kernel (i.e. its size $F \times F$), P is the amount of padding that is used on the boarders, and W and H are the width and height of the feature maps indexed by 1 for the input and 2 for the output [12]. The setting of the hyperparameters is not feasible when the width and height of the output calculated with equation 17 and 18, respectively, doesn't result in an integer, since a result with a fractional component indicates that there is a mismatch between input and output (i.e. the kernel cannot slide neatly across the input's entire width and height) [12]. This is in fact what happens when the input in the aforementioned example was 10×10 instead of 5×5, because without padding and with a 3×3 kernel whose stride length is 2, the output width would be $(W_1 - F - 2P)/S + 1 = (10 - 3 + 0)/2 + 1 = 4.5$ and thus not an integer.

Apart from increasing the stride length, however, there is another way to reduce the resolution of the input which essentially all CNNs employ [4]. The respective operation is called pooling. It reduces the number of parameters and therefore the computational complexity of the network [8]. Just like the convolutional layer, the pooling layer is composed of neurons that are connected to their receptive field in the layer preceding it [8]. Pooling means that the values in the local receptive fields, which determine the output of the respective units in the output layer, are summarized to one value. Most CNNs use max-pooling to simply output the maximum value of the receptive field [8]. Figure 2.3 visualizes

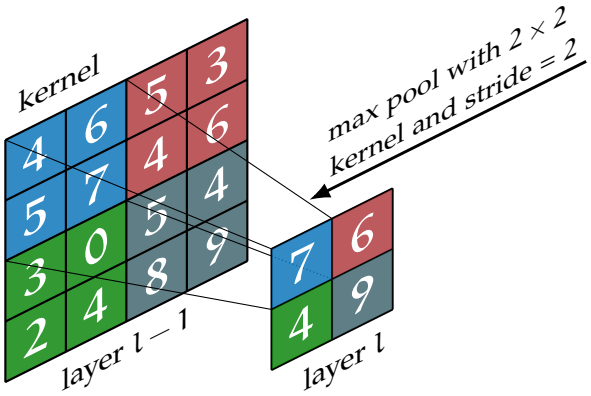


Figure 2.3: A filter of size 2×2 slides with a stride length of 2 across the input in order to downsample the spatial dimension from 4×4 to 2×2 by outputting the maximum value of the local receptive field. This operation is called max pooling.

a simple example of the max pooling operation in which a kernel of size 2×2 with a stride length of 2 downsamples the input layer $l-1$ from 4×4 to 2×2. More generally, the output dimension of the pooling layer can be calculated analogous to the output of the convolutional layer. As padding is usually not applied in a pooling layer, the output dimension $W_2 \times H_2$ is given by

$$W_2 = \frac{W_1 - F}{S} + 1\,, \tag{19}$$

$$H_2 = \frac{H_1 - F}{S} + 1\,, \tag{20}$$

where F denotes the spatial extent of the kernel and S represents the stride length.

With the convolutional and the pooling layer sketched here as well as the fully-connected layer discussed in section 2.1.1, the main components of CNNs have been presented. However, for the sake of simplicity the examples given in this section showed a two-dimensional input (i.e. with a depth of 1), while generally CNNs have to deal with images containing multiple channels such as standard RGB images (i.e. with a depth of 3). In this case, the receptive field and the kernel must have the same depth as the input [12]. In other words, the kernel in a convolutional layer always extends through the input's entire depth [12]. The weighted sum is then basically performed in the same way as shown before, but for each channel in the input and its corresponding depth slice in the kernel separately, before the results for each channel are summed together element-wise to form the final output of the convolution operation [13, 12]. This means that if, for example, only the depth (i.e. the number of channels) of layer $l-1$ and correspondingly the depth of the kernel in figure 2.2 were changed, the dimensionality of the output layer l wouldn't be changed. The depth of the output would still be 1, so the convolutional layer would decrease the number of

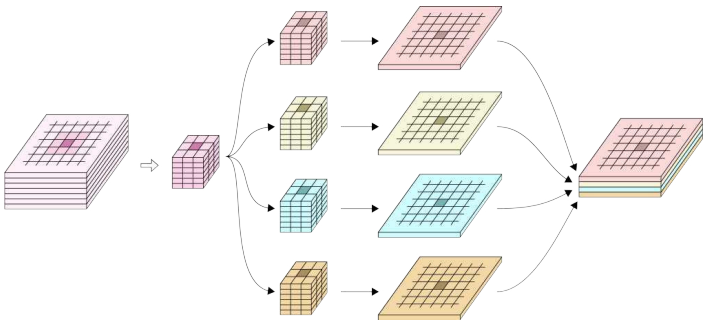


Figure 2.4: Convolution with multiple input and output channels. Each kernel extends through the full depth of the input (i.e. takes all D_1 channels as input). The convolution with each kernel results in one of the D_2 output channels. Since $K = 4$ kernels are applied, the depth of the output is $D_2 = K = 4$. Each kernel has $F \times C_1$ weights with F denoting the spatial extend of the kernel (i.e. its size $K \times K$), so the total number of learnable weights is $F^2 \times D_1 \times D_2$ [3].

channels in this example from 3 to 1. Put more generally, it means that applying one kernel in a convolutional layer always leads to an output feature map with a depth of 1. However, usually multiple kernels (with the depth of the input) are applied in one convolutional layer, which finally leads to the last hyperparameter representing the depth of the output volume. Along with the width W_2 and height H_2 calculated by equation 17 and 18, respectively, the output volume of a convolutional layer is complemented by the depth

$$D_2 = K\,, \tag{21}$$

where K denoted the number of kernels. Usually, the number of kernels can be inferred indirectly from neural network diagrams, as convolutional layer are labeled mostly only with the kernel size (without depth) and the number of output channels [3]. Figure 2.4 shows exemplary a convolution with $K = 4$ kernels with a size of $F = 3$ (i.e. 3×3 kernels) that map D_1 input channels to $D_2 = K = 4$ output channels, which makes a total of $F^2 \times D_1 \times D_2$ learnable weights in this layer. In summary, the convolutional layer

- accepts a tensor of size $W_1 \times H_1 \times D_1$
- requires 4 hyperparameters: the number of filters K, their spatial extend F, the stride S, and the amount of zero padding P
- produces a volume of size $W_2 \times H_2 \times D_2$ computed with equation 17, 18 and 21, respectively, where the d-th channel in the output represents the result of the convolution with the d-th kernel over the input
- introduces (with parameter sharing) $D_1 F^2 K$ learnable weights and K biases

Just like the convolution operation, the pooling operation was demonstrated for an input with a depth of 1. Unlike the convolutional layer, however, the pooling layer operates independently on every depth slice (or channel) in the input, so it downsamples only the height and width, while the depth of the output is always equal to the depth of its input [12]. Therefore, in general, the pooling layer

- accepts a tensor of size $W_1 \times H_1 \times D_1$
- requires 2 hyperparameters: the spatial extend F and the stride S
- produces a volume of size $W_2 \times H_2 \times D_2$, where the hight and width results from equation 19 and 20, respectively, and the depth D_2 is equal to D_1

The functioning of a CNN can now be summarized as follows: In a convolutional layer, a filter matrix of a size equivalent to the receptive field, whose elements are referred to as weights, is convolved with the values of the receptive field to determine outputs of the convolutional layer respectively the outputs of its corresponding neurons [8, 12, 14]. A pooling layer summarizes the values of the local receptive field in order to decrease the complexity of the network. With an interleave of multiple layers, which is the definition of deep learning, CNNs basically construct a feature hierarchy of its input [8, 14]. By adding fully connected layers to the interleave of convolutional and pooling layers, non-linear combinations can be learned from the constructed feature hierarchy in a supervised manner, i.e. with labeled input data [14]. More specifically, CNNs can be optimized on a loss function (e.g. MSE or negative cross-entropy) using the SGD algorithm introduced in section 2.1.1, where the equations of back-propagation and the parameter updates must be adjusted to the computational elements (i.e. convolution and pooling) of the layers in the CNN. Simply put, this enables a CNNs to adapted to different visual tasks [14]. According to the respective task, the final layer contains activation functions to predict a conditional probability for each neuron [14]. In section 2.1.1, these neurons were summarized by the vector $\mathbf{a}^{(L)}$. As an example, the probability (also called score or confidence) computed by an object detection or instance segmentation system is, simply put, the predicted probability of whether or not a bounding box or an instance mask is actually the predicted object [15].

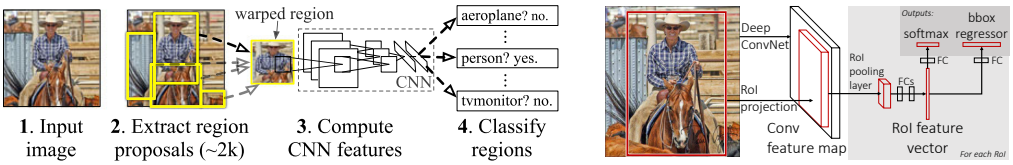
Despite the relatively small number of different layers, building a CNN is more complicated than simply stacking multiple layers, especially when it comes to difficult tasks such as object detection and instance segmentation. These tasks require numerous additional elements (such as region proposal generation, bounding box regression, etc.) that are not sketched in this section. Therefore, section 2.1.3 and 2.1.4 discusses these two vision tasks before section 2.1.5 presents state-of-the-art CNN architectures.

2.1.3 Object Detection

Per definition object detection systems perform both classification and localization of objects in input images. Assigning a proper class label to each foreground object is the solution of the classification problem, whereas accurate assignments of bounding boxes to these objects refers to the solution of the localization problem [2]. Consequently, object detectors aim to assign a class label (to solve the recognization or classification problem) and a rectangular bounding box (to solve the localization problem) to each object in an image [14]. Additionally, as noted in section 2.1.2, CNN-based object detectors predict the probability of whether or not a detection actually is the respective object and score each detection bounding box accordingly.

Modern object detection systems are usually divided into two categories, one of which is a two-stage method and the second one is a single-stage approach [16]. Algorithms following the former approach, which are also referred to as region proposal based frameworks, generate several region proposals in the first stage in order to perform region-wise regression of features and classifications in object categories in the second stage [16, 2, 3]. Single-stage object detection systems, on the other hand, use a unified framework that delivers recognization and localization results directly [14]. It can generally be stated that two-stage object detection systems perform well with respect to localization and recognization accuracy, whereas single-stage detectors, although they typically lag behind in terms of accuracy compared to their two-stage counterparts, are computationally efficient and thus reach high inference speed [16, 2].

Indeed, the limitations in detection accuracy of single-stage detectors have been addressed and detectors with results comparable to two-stage detectors were proposed [2]. However, with many of the recently proposed object detectors following the two-stage approach, the region proposal based framework has become predominant in the past years [2]. In fact, many modern object detection systems extend the region based framework Faster R-CNN proposed by Ren et al. in 2016 in order to address various problems of detail, making it the cornerstone of modern object detection [17, 2]. This goes well beyond object detection as even modern instance segmentation frameworks are based on the Faster R-CNN framework [18]. Cascade Mask R-CNN, for instance, which is used in this work for both object detection and instance segmentation, is a multi-stage extension of the Faster R-CNN architecture with state-of-the-art performance in general object detection and instance segmentation tasks [2, 6]. Also HTC, another architecture selected for evaluation in this work, pursued the idea behind Cascade R-CNN evolving in a different Cascade architecture that is not directly a Faster R-CNN extension but still follows the region proposal based approach [2, 1]. Both Cascade Mask R-CNN and HTC will be described in section 2.1.5. Previously, however, the basis will be provided with an explanation of region based frameworks from



(a) R-CNN (b) Fast R-CNN

Figure 2.5: (a) R-CNN takes an input image (1) and performs object detection in 3 stages: (2) region proposal generation with selective search, (3) feature extraction for each region proposal with a CNN, (4) final classification and bounding box regression [19]. (b) Fast R-CNN performs feature extraction with a CNN on the entire image before extracting fixed-size feature vectors for individual region proposals. While region proposals are generated by methods such as selective search, the feature vectors are extracted with a RoI pooling layer. Finally, after passing the feature vectors through fully connected layers, two sibling output layers perform bounding box regression and softmax classification, respectively [21].

the first approach to Faster R-CNN, and up to Mask R-CNN, from which the latter will be introduced in section 2.1.4 as it provides (with an additional mask branch for Faster R-CNN) a framework for instance segmentation.

2.1.3.1 R-CNN

The approach of combining region proposals with CNNs was introduced by Girshick et al. in 2014 [19]. As shown in figure 2.5 a, the proposed architecture, called R-CNN, consists of three modules through which input images are fed. In the first module, a method called selective search is used to generate region proposals of arbitrary sizes and aspect ratios, which have to be rescaled (warped), since the fully connected layers in the CNN model represented by the subsequent module need a fixed-size input [16, 20]. To this end, each so-called Region of Interest (RoI) is warped to a 227×227 image [19]. Thereafter, the fixed-size region proposals are fed separately into a CNN with five convolutional and two fully-connected layers in order to extract features as described in section 2.1.2 and 2.1.1, respectively [19]. Finally, in the last module, the features from each region, which are represented by 4096-dimensional feature vectors, are scored by pre-trained class-specific Support Vector Machines (SVMs) (i.e. classified), adjusted by bounding box regression and filtered with non-maximum suppression (NMS) in order to produce final bounding boxes, class labels and scores for obtained object locations and classes, respectively [19].

2.1.3.2 Fast R-CNN

R-CNN improved the object detection performance compared to previous models. However, its architecture is very time-consuming and inaccurate compared to modern detectors [14]. The mentioned feature extraction for each RoI with a CNN is very expensive in time and warping regions reduces recognition accuracy [21, 22]. In a time of active research, various region proposal based frameworks emerged and improved both accuracy and speed.

One of these frameworks, called Fast R-CNN, uses convolutional layers in the first stage to produce feature maps for the whole image all at once [14]. Thereafter, a fixed size feature vector is extracted for each region proposal regardless of the RoI's size and aspect ratio [16]. The region proposals are, just like in R-CNN, produced with selective search, and the extraction of fixed-length feature vectors from each RoI in the convolutional feature map is performed by a so-called RoI pooling layer [14, 21]. Finally, the fixed-size feature vectors are fed into fully connected layers before two sibling output layers perform bounding box regression and classification, respectively [14]. Rather than utilizing SVMs for classification (as R-CNN does), Fast R-CNN makes use of softmax classification, which together with its sibling output layer for bounding box regression is part of the architecture's two heads [3]. In fact, these two separate heads for both tasks require two different loss functions for training [3]. Both of them are summarized to a so-called multi-task loss in order to be able to jointly train classification and bounding box regression [14]. The loss is defined as

$$\mathcal{L}(p, u, t^u, v) = \mathcal{L}_{cls}(p, u) + \lambda[u \geqslant 1]\mathcal{L}_{loc}(t^u, v)\,, \tag{22}$$

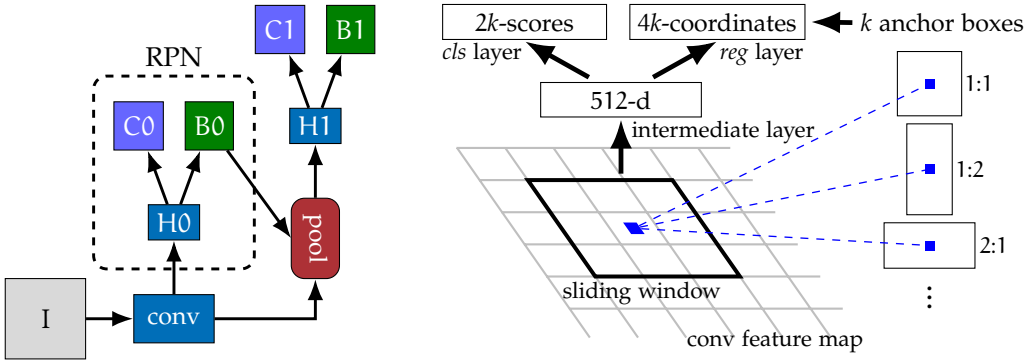
where $\mathcal{L}_{cls}(p, u)$ is the classification loss and $\mathcal{L}_{loc}(t^u, v)$ is the localization or regression loss. The classification loss is based on the ground truth class u and the predicted probability distribution $p = (p_0, ..., p_{C-1})$. Here, the probability distribution predicts the respective class with C outputs (for each of the C categories) represented by neurons in the last fully-connected layer of the classification head [14]. During the training, a RoI (extracted by the RoI pooling layer) is labeled with a ground truth class u when the RoI together with a ground truth bounding box exceed a certain Intersection over Union (IoU) threshold (for a detailed description of the IoU see section 2.3.1) [21]. This means that $u \geqslant 1$ suggests that a RoI matches a ground truth bounding box with respect to the IoU threshold, and the Iverson bracket indicator function $[u \geqslant 1]$ evaluates to 1. Conversely, the term $[u \geqslant 1]$ evaluates to 0 if the RoI doesn't match any ground truth bounding box with respect to the IoU threshold (i.e. when $u = 0 =$ background). In other words, the term $[u \geqslant 1]\mathcal{L}_{loc}$ basically selects a set of positive RoI examples to train the regressor (when $[u \geqslant 1]$ is equal to 1) and omits background RoIs (when $[u \geqslant 1]$ is equal to 0) [14, 2]. The regression loss itself, $\mathcal{L}_{loc}(t^u, v)$, is defined over

the predicted offsets $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$ and the ground truth bounding-box regression targets $v = (v_x, v_y, v_w, v_h)$ [14]. Finally, neural networks with multiple heads and thus outputs (here for classification and regression) must be carefully balanced, which is the purpose of the factor λ [3, 21]. Fast R-CNN can be trained end-to-end by optimizing, with the sole exception of the region proposal generation, all parameters with the multi-task loss in equation 22 [14]. Its architecture is shown in figure 2.5 b.

In summary, by interchanging the CNN and region extraction stages, Fast R-CNN improved the performance compared to the first R-CNN architecture in which the region extraction happens first [3]. This improvement is most obvious in terms of speed as the convolutional network in R-CNN must process each region proposal individually, whereas Fast R-CNN applies the CNN in the first stage in order to generate feature maps for the entire image only once. However, although interchanging the stages in the aforementioned way resulted in considerable speed improvements, Fast R-CNN relies, just like the first R-CNN architecture, on methods such as selective search for generating region proposals [14]. This is time-consuming and thus an obstacle on the path to real-time object detection [14].

2.1.3.3 Faster R-CNN

Ren et al. addressed this problem and observed that feature maps extracted by a CNN can also be used to generate region proposals [17]. They introduced a Region Proposal Network (RPN), combined it with the Fast R-CNN detector and called the composed system Faster R-CNN. More specifically, a deep CNN such as VGG-16, which consists of 13 convolutional layers, 3 fully-connected layers and 4 max pooling layers takes an input image and extracts feature maps as explained in section 2.1.1 and 2.1.2 [17, 23]. The RPN operates in a sliding-window manner on a specific convolutional layer of the CNN with the layers preceding it [14]. It contains two sibling fully-connected layers for classification (*cls* layer) and box regression (*reg* layer) to generate a set of rectangular region proposals and to score these proposals based on the predicted probabilities of whether or not they contain an object [14]. Since the fully-image features extracted in the convolutional layers are not only used by the RPN but also shared with the Fast R-CNN detector in the second stage, region proposals are computed in a nearly cost-free way [17]. This is a notable difference to the slow selective search method leveraged by the previous models for generating RoI's. Finally, both the relevant RoI's produced by the RPN and the feature maps produced by the backbone CNN are used by the Fast R-CNN detector, which in turn evaluates each region proposal in the above-mentioned manner (i.e. with a RoI pooling layer, a softmax classifier and bounding box regression) to output bounding boxes and scores (or confidences) for detected class-specific objects [17]. The Faster



(a) Faster R-CNN architecture (b) Region Proposal Network (RPN)

Figure 2.6: (a) The Faster R-CNN architecture has a shared backbone CNN "conv" for feature generation. A RPN operates on a specific convolutional layer in "conv" and produces RoI's. The RoI's along with the features from "conv" are evaluated by Fast R-CNN with a RoI pooling layer "pool" (which is not to be confused with the pooling operation described in section 2.1.2) and fully connected layers represented by the head "H1" as well as two sibling output layers for classification ("C1") and bounding box regression ("B1"). (b) The RPN operates in a sliding-window fashion with a convolutional layer that extracts 512-d feature vectors and that simultaneously predicts k region proposals represented by anchor boxes with different scales and aspect ratios centered at each sliding window position. The region proposals are refined and classified by a *reg* and *cls* layer, respectively. The RPN categorizes object only into foreground and background without taking into account to which class the object belongs.

R-CNN architecture and its RPN are shown in figure 2.6 a and 2.6 b, respectively.

As mentioned before, both architectures used in this work are related to the Faster R-CNN architecture. Cascade Mask R-CNN is a multi-stage extension of Faster R-CNN and HTC is another Cascade architecture that uses a RPN for region proposal generation. Consequently, the architecture is described in more detail below. Although Faster R-CNN is now often used with different CNNs such as ResNet-101 as backbone, the description will be based on the VGG-16 as proposed by Ren et al. in 2016 [24, 17].

The backbone network VGG-16, through which input images are passed in the first stage, has 13 convolutional layers and 4 max pooling layers [23]. Each convolutional layer operates with 3×3 kernels, a stride length of 1, and the padding is 1 pixel, which according to equation 17 and 18 preserves the resolution after each convolution. Only the number of kernels in each layer differs and ranges from 64 to 512. The four pooling layers operate with a 2×2 kernel and a stride length of 2, so each pooling operation, which perform max pooling, cut the resolution in half based on equation 19 and 20. The published paper states

that the RPN operates at the last "shared" convolutional layer of VGG-16, which according to the original implementation is actually its 13-th convolutional layer and thus the last one of the entire backbone network [17, 25]. This specific layer has 512 output channels and the output feature map's spatial dimension is 1/16 of the original input image as it is preceded by all four aforementioned max pooling layers that downsample the input by a factor of 2 each time [17, 25].

The intermediate layer H0 of the RPN is also a convolutional layer with a 3×3 kernel, a stride length of 1, and 512 output channels, which uses padding of 1 pixel so that the spatial dimension remains unchanged. This means that according to the explanation in section 2.1.2, the intermediate layer of the RPN contains 512 kernels that extract a 512-dimensional feature vector at each spatial position of the sliding window in order to form all 512 output channels. To increase non-linearity, ReLU is applied at the output of the convolutional layer [14]. Figure 2.6 b shows one position of the so-called mini-network that slides across the feature map outputted by the last convolutional layer of VGG-16, which extracts at this specific location a 512-d feature vector [17, 25].

In addition to the extraction of the 512-d feature vector at every position of the sliding window, the mini-network predicts k region proposals called anchors with different scales and aspect ratios that are centered in the 3×3 sliding window. These anchor boxes are basically projected into the original image, where they are centered at the point of the image that corresponds to the point in the feature map (marked in blue in figure 2.6 b). Now, due to the fact that the feature map, on which the RPN operates, is downsampled by a factor of 16 (through the four max pooling layers), the total stride is also 16, meaning that one pixel to the right (or down) on the feature map represents a stride of 16 pixels on the original image. This means subsequently that the center of all k anchors for every sliding window position, which are based on the feature map but actually represent regions in the original image, are located with a distance of 16 pixels to the center of their neighboring anchors. The proposed system applies anchor boxes with three aspect ratios and three scales, which makes 9 anchors at every sliding window position and thus a total of WHk anchors on a $W \times H$ feature map. To put this into perspective, a typical 1000×600 image would be downsampled by a factor of 16 so that the spatial dimension of the feature map after the final convolutional layer in VGG-16 equals approximately 60×40. Consequently, there would be approximately $20k(\approx 60 \cdot 40 \cdot 9)$ anchors projected into the original image. In the proposed Faster R-CNN architecture, the scales of the anchor boxes are 128^2, 256^2 and 512^2 pixels and each scaled anchor is projected with the aspect ratios $1:1$, $1:2$ and $2:1$ into the original image [17, 25].

The output of the intermediate layer H0 is then passed to two sibling layers for box regression and classification. The first one, called *cls* layer (C0), is a convolutional layer with a 1×1 kernel, a stride length of 1 and no padding, which

maps the 512 input channels from the intermediate layer to $2k$ (i.e. $2 \cdot 9 = 18$) output channels. With this, it produces at each spatial window position $2k$ scores representing for each anchor centered at this particular location the probability of whether it actually is an object (first score) or no object and thus background (second score). The other sibling layer, called *reg* layer (B0), is also a convolutional layer with a 1×1 kernel, a stride length of 1 and no padding, but it maps the 512 input channels from the intermediate layer to $4k$ (i.e. $4 \cdot 9 = 36$) channels. These channels represent 4 regression coordinates for each one of the k anchor boxes at each sliding window position, which correct the original coordinates of the respective anchor boxes before passing them to the RoI pooling layer [17, 25].

In summary, the RPN is a fully convolutional network that outputs WHk region proposals, which along with the feature map from the VGG-16 network are passed to the Fast R-CNN detector for final evaluation towards true bounding boxes (i.e. classification and bounding box regression) as explained before. In figure 2.6 a, the parts of Fast R-CNN are represented by "pool" for the RoI pooling layer, H1 for the fully connected layers, as well as C1 and B1 for the architecture's two heads (bounding box regression and classification) [17].

The entire Faster R-CNN network contains four losses. Two of them represent the classification loss and regression loss inside the RPN for a kind of class agnostic object detection, i.e. for a localization of proposals (or anchors) and their classification in "foreground" and "background" without taking the specific category into account [26]. It is defined similarly to the loss in equation 22 by

$$\mathcal{L}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* \mathcal{L}_{reg}(t_i, t_i^*), \tag{23}$$

where p_i is the predicted probably of whether the i-th anchor is an object. The term p_i^* is basically an assignment in positive or negative anchors based on the IoU of anchor i with any ground truth bounding box. More specifically, p_i^* is 1 for a positive or 0 for a negative assignment and, analogous to u in equation 22, selects the set of samples (i.e. anchors) used to train the regressor. To this end, it activates the regression loss by the term $p_i^* \mathcal{L}_{reg}$ for positive anchors ($p_i^* = 1$) and disables it otherwise ($p_i^* = 0$). The regression loss $\mathcal{L}_{reg}$ itself is based on four predicted bounding box coordinates represented by t_i and four coordinates of the ground truth bounding box t_i^* associated with a positive anchor. Finally, the terms for classification and regression are normalized by N_{cls} and N_{reg}, respectively, and the factor λ is analogous to the loss in equation 23 applied for balancing the two individual loss functions [17].

The other two losses are represented by the multi-task loss from the Fast R-CNN detector in the second stage for class specific object detection and thus defined by equation 22. By jointly optimizing all four losses and thus all parameters including those for the region proposal generation, Faster R-CNN can be trained

end-to-end with back-propagation and SGD according to section 2.1.1 and 2.1.2 [14, 26]. This is an additional improvement compared to the Fast R-CNN network, which can only optimize the parameters that aren't related to region proposals. On a final note it must be stated that based on the explanation above, plenty of region proposals (i.e. anchors) cross image boundaries (e.g. those anchors that are projected from the boarders of the feature map into the original image). In the training process, these anchors are ignored, so they won't contribute to the loss [17].

In conclusion, the developments of region proposal based object detection in the past decade can be summarized as follows. With respect to the test time, the first architecture, R-CNN, took about 50 seconds for one image. Compared to this, Fast R-CNN and Faster R-CNN achieved a 25× and 250× speedup, respectively [26]. With respect to the training, the process went from a multi stage pipeline in R-CNN, in which different parts of the network had to be trained separately, to the Faster R-CNN network, which can really be trained end-to-end. Today, Faster R-CNN is mostly combined with the 101 layer deep network ResNet-101 or other advances in backbone architectures as they outperformed the original implementation with VGG-16 [24]. It is still state-of-the-art and, again, extended by many modern detectors to address various problems of detail. Mask R-CNN is one example. It will be introduced in section 2.1.4.

2.1.4 Instance Segmentation

Instance segmentation extends the object detection task in terms of accuracy by producing pixel-accurate masks for visible regions of the objects rather than drawing bounding boxes around them [3]. A breakthrough in instance segmentation came with the introduction of Mask R-CNN by He et al. in 2017 [3, 18]. It uses the same region proposal network as Faster R-CNN and adds an additional branch for predicting the object mask in parallel to the existing branch for classification and bounding box regression [18]. Figure 2.7 shows the architecture, which is essentially the same architecture as Faster R-CNN in figure 2.6 a. Only the branch S for predicting the object mask is added [2].

Without going into detail, the mask branch is a small FCN applied to each RoI and produces binary masks for localized objects [18]. Consequently, it assigns a pixel-level semantic class label to each pixel belonging to the object localized within the respective RoI. A notable difference between Mask R-CNN and Faster R-CNN architecture apart from the mask branch S regards the pooling layer that extracts fixed size feature vectors from the region proposals [18, 21]. The RoI pooling layer in Faster R-CNN (and Fast R-CNN) was designed for bounding boxes and performs spatial quantization of features that are too coarse for pixel-to-pixel alignments [18]. He et al. introduced a layer called RoIAlign that preserved the exact spatial locations of features [18]. Although replacing the RoI pooling

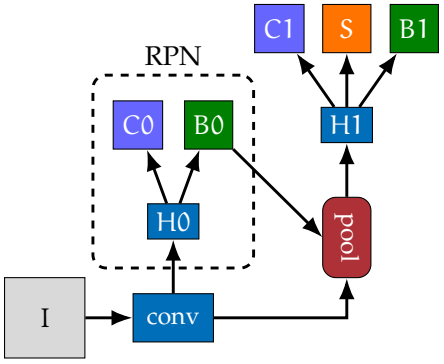


Figure 2.7: The Mask R-CNN architecture complements the Faster R-CNN architecture by adding the mask branch "S" for instance segmentation in parallel to the existing branch for classification "C1" and bounding box regression "B1".

layer with RoIAlign is a small change, it has a significant effect on the mask accuracy. It resulted in a relative improvement by 10% to 50% [18].

Mask R-CNN has multiple heads and requires, like Fast R-CNN and Faster R-CNN, several losses. Straightforwardly, a loss for the mask branch $\mathcal{L}_{mask}$ is added to the multi-task loss defined by equation 22. Therefore, the loss for Mask R-CNN, whose individual terms must be balanced just like that of the multi-task losses described in section 2.1.3, is formally defined as

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{reg} + \mathcal{L}_{mask}\,. \tag{24}$$

Analogous to the regression loss in Faster R-CNN (and Fast R-CNN), the mask loss $\mathcal{L}_{mask}$ is activated when the RoIs is considered positive. And the classification in positive or negative itself is based on whether a RoI matches a ground truth box with respect to a certain IoU threshold. Consequently, the set of samples used to train the mask loss is chosen analogously to that for training the regression loss. In mathematical terms, the selection may thus be defined analogously to the regression loss in equation 22 with the term $[u \geqslant 1]$, where u denotes the ground truth class with which the respective RoI is labeled (i.e. $[u \geqslant 1] = 0$ if $u = 0 =$ background and $[u \geqslant 1] = 1$ if $u \geqslant 1 =$ foreground category) [18].

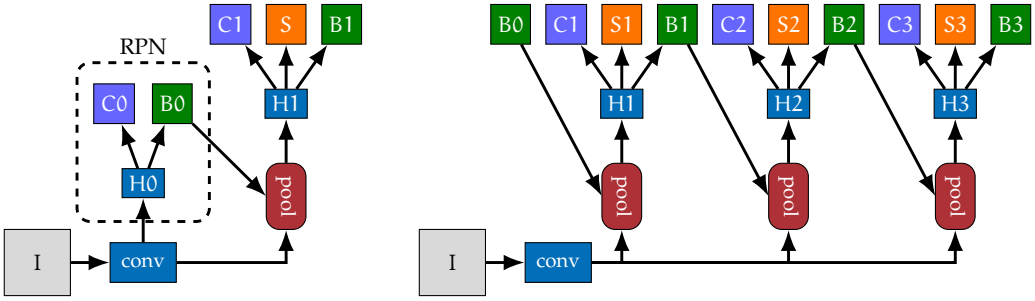
Mask R-CNN outputs for each RoI K binary masks with the size $m \times m$, one for each of the K classes. However, for an RoI associated with class u (provided that $[u \geqslant 1] = 0$), the loss is only defined on the u-th mask (i.e. the mask outputted for the respective class), whereas the remaining outputs do not contribute to the loss. The loss $\mathcal{L}_{mask}$ is then defined as the the average binary cross-entropy [18]. Since Mask R-CNN contains the same RPN as Faster R-CNN, two additional losses are used for the RPN optimization. They are defined by the multi-task loss in equation 23 [18].

2.1.5 State-of-the-art Architectures

This section presents state-of-the-art Computer Vision architectures with the focus on the region proposal based approach. With Faster R-CNN and Mask R-CNN, two modern architectures are already described. This section adds two more architectures, both of which are region proposal based Cascade (or multi-stage) architectures. The first multi-stage architecture was introduced by Cai et al. in 2018 under the name Cascade R-CNN [27]. Later, they combined the architecture with a mask branch for instance segmentation and called it Cascade Mask R-CNN [2]. With the first proposal they already reported improvements in object detection over their two-stage counterpart Faster R-CNN, but the simple combination of their multi-stage architecture with a mask branch later showed improvements in instance segmentation over Mask R-CNN as well [2]. The second architecture, HTC, focused on the question how the Cascade architecture can be introduced to instance segmentation in a more sophisticated way than simply combining Cascade R-CNN and Mask R-CNN [1]. Nevertheless, both of these mult-stage architectures showed strong results in both instance segmentation and object detection.

The idea behind the Cascade architecture proposed by Cai et al. in 2018 is to sequentially increase the quality of proposals in a region proposal based architecture, as they observed that the quality of the detector directly correlates with the quality of object (or region) proposals [2]. Here, the quality is defined by the IoU between the candidate and ground truth bounding boxes [2]. The overlap criterion represented by the IoU will be described with respect to evaluation methodologies in section 2.3.1, but the IoU is also used to train object detection systems. As shown with equation 22 and 23 in section 2.1.3, in modern region-proposal based architectures, this holds true for both optimizing the final detection task and the RPN. Regarding the RPN, equation 23 shows that in the *cls* head ("C0") the IoU threshold is used to classify proposals (or anchors) in the categories "positive" (object) and "negative" (background), and in the *reg* head ("B0"), although the bounding box regression task doesn't directly need a definition in positive and negative examples, the IoU threshold is required to select the set of samples used to train the regressor [2]. And this is essentially where the observations by Cai et al. in 2018 begins [2].

The observations eventually resulted in Cascade Mask R-CNN shown in figure 2.8 b and can be summarized as follows. Setting the IoU threshold for the RPN training up obviously increases the quality of object proposals (i.e higher IoU between proposal and ground truth). In theory, this also increases the object detection quality because as mentioned above there happens to be a match between the quality of the detector and that of the proposal. However, the RPN tends to produce object proposals that are imbalanced towards low quality, which has a dramatic effect on the absolute number of proposals during training



(a) Mask R-CNN (b) Cascade Mask R-CNN

Figure 2.8: Cascade Mask R-CNN represents a sequence of object detectors trained with increased IoU thresholds. They use object proposals from the previous model with a lower IoU threshold in order to produce object proposals of better quality for the subsequent model. "B0" represents the proposals from the RPN in both Mask R-CNN and Cascade Mask R-CNN.

when the IoU threshold is high. In other words, using larger IoU thresholds exponentially reduces the number of proposals produced by the RPN, which in turn decreases the number of positive examples for training the detection head "H1" in the second stage. This means that with a large IoU threshold, the detection head basically sees the same training examples over and over again. Subsequently, since neural network are very example intensive, the model can't generalize well and overfits these particular training examples that are left over [2].

A final observation which intuitively explains the cascade architecture is that a bounding box regressor trained for a certain IoU threshold tends to produce bounding boxes of higher IoUs. The idea of Cascade R-CNN is to start with an object detector trained with a low IoU threshold and use its output as object proposals for training a second object detector with an increased IoU threshold. This process is basically repeated to sequentially use outputs of a detector trained with a lower IoU threshold as proposals for training a detector with the next higher IoU threshold. Through this process, the quality of region proposals outputted by the RPN in the first stage increases with each subsequent stage without risking to reducing the number of training examples. Consequently, a higher object detection accuracy is achieved by avoiding the aforementioned overfitting [2].

Finally a mask branch "S" is added to Cascade R-CNN. Since the multi-stage architecture contains multiple detection branches, Cai et al. tested various strategies. One of them is shown in figure 2.8 b where the segmentation branch is added in every stage. In other strategies they implemented only one segmentation branch. However, regardless of the training strategy, at inference time the segmentation mask is produced in parallel to the last detection stage [2].

The improvements for instance segmentation achieved by Cascade Mask

R-CNN are, unlike the obvious improvements in object detection, non-trivial because the information flow happens only with respect to bounding boxes [2, 1]. Training the segmentation branch happens in the respective stages in parallel to the bounding box regression, but the increased quality of object proposals in later stages doesn't necessarily help pixel-wise operations performed by the segmentation branch [2]. This is in fact precisely why Cai et al. tried the various aforementioned strategies with respect to the segmentation branch [2]. HTC proposed another Cascade architecture to address this problem. Without going into detail, rather than performing bounding box regression and mask prediction in parallel, HTC interleaves these tasks in order to improve the information flow [1]. With this, HTC achieved strong performance in both instance segmentation and object detection.

2.2 Autonomous Driving Datasets

Datasets have played a profound role in the development of computer vision [28]. The whole progress presented in section 2.1.3, 2.1.4 and 2.1.5 wouldn't have been possible without high-quality datasets, as all CNN-based Computer Vision systems rely on supervised learning. The classical paradigm of supervised learning is to partition the dataset into three statistically independent sets: training, validation, and test set [8]. In fact, even though some datasets are released only with a test and a training set, it is still necessary to subdivision the latter into two separate sets for training and validation [28]. The intent here is to leave the decision about the specific partition to users (i.e. regarding the training-validation ratio) [28]. The ultimate purpose of partitioning the dataset into three independently and identically distributed sets is to ensure that the algorithm generalizes well during the training and thus shows a good performance on unseen data. The following paragraph elaborates on this learning paradigm.

As its name implies, the training set, which contains most of the data in the dataset, is used to train a deep learning model. Simply put, it is the only data in the dataset to which the algorithm has direct access. Section 2.1.1 describes the training process in detail using a FCN as an example. Indeed, the process is mathematically more complicated with state-of-the-art CNNs. These neural networks incorporate additional elements such as convolutional and pooling layer (see section 2.1.2) as well as various techniques and specific loss functions for computer vision tasks such as object detection and instance segmentation (see section 2.1.3 to 2.1.5). The main concepts of SGD and back-propagation, however, are essentially the same as in FCNs. And regardless of which computational elements are used, plenty of choices are to be made prior to the training of neural networks. Section 2.1.2 discussed parameter such as the stride and the kernel size that need to be defined for convolutional and pooling layer. Additional choices

such as the learning rate, η, and the type of loss function, $\mathcal{L}$, were described in section 2.1.1. All of these so-called hyper-parameter have in common that they can't actually be learned. Instead, they must be manually defined. To this end, the validation set basically provides (for the algorithm unseen) data to figure out how to set hyper-parameters properly [7]. More specifically, since choices of hyper-parameters are not necessarily trivial, usually different sets of hyper-parameters are tried during the training process and evaluated with the validation set. However, picking the set of hyper-parameters that results in the best performance on the validation set might, in the worst case, only be the set that accidentally causes the algorithm to perform well on the validation set [29]. In fact, multiple settings are tried until the best performance is reached, so the final performance on the validation set is no longer representative of the performance on unseen data. Therefore, the test set is used after the whole training is completed to provide a final assessment of the algorithm's performance on unseen data.

Partitioning the dataset is essential for all Computer Vision dataset. What distinguishes Computer Vision datasets, however, are the specific tasks they are used for. In this work the focus is on object detection and instance segmentation. The former, object detection, requires datasets with class-specific bounding box coordinates for each object, whereas the latter, instance segmentation, requires class-specific polygons or binary masks for each object. Both tasks will be examined in the field of Autonomous Driving. Due to the growing research interest in self-driving cars in the past decades, numerous datasets with driving scenes captured in urban, non-city and agricultural environments have been published. Table 2.1 lists some Autonomous Driving datasets that are worth mentioning.

Apart from the obvious improvements through the architectural developments of deep learning algorithms (see section 2.1.3 to 2.1.5), the progress in Computer Vision can primarily be attributed to datasets with respect to their high-quality data for training. However, datasets also galvanized research progress in Computer Vision from another point of view. Many datasets such as Caltech, KITTI, and COCO include online rankings, which have led to competitions between various research teams and thus contributed to the rapid progress in Computer Vision in the last decade [30, 31, 37]. Particularly important for the research process in Computer Vision in the last decade is the MS COCO dataset. The first version of the dataset was released in 2014 and coincided with the shift in Computer Vision [3]. This alludes not only to the introduction of the first Region-based Convolutional Neural Network (R-CNN) for object detection in 2014 (see section 2.1.3), but also to deep networks for instance segmentation. It is a general recognition dataset containing 300,000 images with annotations for both object detection and instance segmentation for 80 classes with an average

Dataset	Published	# Images	# Classes	# Objects	Resolution
Caltech [30]	2009	250,000	1	350,000	640×480
KITTI [31]	2012	14,999	8	80,256	1392×512
NREC [28]	2017	95,924	1	76,662	720×480
CityScapes [32]	2016	5000	8	65,400	2048×1024
CityPersons [33]	2017	5000	1	35,016	2048×1024
EuroCity Persons [34]	2019	238,200	2	47,335	1920×1024
Waymo [35]	2020	1.15M	3	9.9M	1280×1920
BDD100K [36]	2020	100,000	11	1,84M	1280×720

Table 2.1: Autonomous Driving Datasets. Caltech, NREC, CityPersons, and EuroCity Persons are person detection datasets with one (person) or two (pedestrian, rider) classes. In addition to bounding boxes coordinates, BDD100K provides annotations for semantic segmentation. Cityscapes and CityPersons provide ground truth data for instance level semantic segmentations. Waymo is not only a Computer Vision but also 3D LiDAR dataset.

of 7 instances per image [14]. Up until today, most Computer Vision Systems evaluate their performance on the MS COCO dataset [3]. In fact, the MS COCO dataset is not only worth to mention due to its prevalence in Computer Vision in general, but also in the context of Autonomous Driving. Although it is not a dataset specifically for Autonomous Driving, it contains a considerable amount of street scenes and a number of traffic participants per image that is comparable to that in Autonomous Driving datasets such as CityScapes [32].

On a final note it must be stated that quality of datasets depends on how well they depict the reality. This is in fact the reason why datasets are often compared not only with respect to the total number of images and object instances but also based on the diversity of scenes. In Autonomous Driving datasets, for instance, images may be captured in urban environments or non-city environments such as highways at different locations, at different times of the day, and in different weather conditions. In summary, not only the aforementioned training paradigm with training, validation, and test sets, but also the data itself must ensure that the algorithms generalize well, or, conversely, don't overfit to the idiosyncrasies of the particular data in the dataset [28]. Various common methodologies that are used for the final evaluation of object detection and instance segmentation systems will be explained in section 2.3.

2.3 Evaluation Methodology

In section 2.1, modern CNN-based Computer Vision systems are described with respect to their functioning. Their main components are described in section 2.1.1 and 2.1.2 and some architectures for both object detection and instance segmentation are presented in section 2.1.3 to 2.1.5. Moreover, it is described which individual loss functions the respective algorithms use for training. All of these loss functions (defined by equation 22, 23 and 24) indicate that the particular metric modern object detection and instance segmentation systems try to optimize is the IoU, which is commonly used to evaluate the accuracy of both localizations and instance segmentations. As shown in section 2.1.3 and 2.1.4, the IoU is used by all of these loss functions to classify proposals into negatives (background) or positives (class agnostic in case of the RPN and class specific in case of the final *cls*, *reg*, and *mask* heads). Section 2.3.1 provides a more detailed description about the overlap criterion as provided before. Nevertheless, with respect to the overall performance evaluation, the IoU is only used to classify individual detections or instance masks in True Positives (TPs) or False Positives (FPs). It is basically only the first step in the evaluation of the overall performance. The second step is presented in section 2.3.2 by providing an explanation of common evaluation metrics used for both object detection and instance segmentation.

2.3.1 Overlap Criterion

The IoU is also known as the Jaccard index or Jaccard similarity coefficient [3]. In the context of Object Detection, it computes the match between a detection bounding box (referred to as $\mathrm{BB}_{\mathrm{dt}}$) and a labeled ground truth bounding box (referred to as $\mathrm{BB}_{\mathrm{gt}}$)

$$\mathrm{IoU} = \frac{\mathrm{BB}_{\mathrm{gt}} \cap \mathrm{BB}_{\mathrm{dt}}}{\mathrm{BB}_{\mathrm{gt}} \cup \mathrm{BB}_{\mathrm{dt}}}, \tag{25}$$

where the numerator is the intersection of $\mathrm{BB}_{\mathrm{dt}}$ and $\mathrm{BB}_{\mathrm{gt}}$ (i.e. the area of overlap of both bounding boxes) and the denominator is the union (i.e. the total area of both bounding boxes combined). Figure 2.9 visualizes the IoU with a schematic formula. With respect to instance segmentation, the intersection over union is computed by

$$\mathrm{IoU}_{\mathrm{BM}} = \frac{\sum \left(\mathrm{px}(\mathrm{BM}_{\mathrm{gt}}) \wedge \mathrm{px}(\mathrm{BM}_{\mathrm{dt}}) \right)}{\sum \left(\mathrm{px}(\mathrm{BM}_{\mathrm{gt}}) \vee \mathrm{px}(\mathrm{BM}_{\mathrm{dt}}) \right)}, \tag{26}$$

where $\mathrm{BM}_{\mathrm{gt}}$ and $\mathrm{BM}_{\mathrm{dt}}$ denote the ground truth and detection mask, respectively. The numerator is the intersection computed by the sum over all pixels that belong to both binary masks (i.e. to both $\mathrm{BM}_{\mathrm{gt}}$ and $\mathrm{BM}_{\mathrm{dt}}$). The denominator is the union calculated by the sum over all pixels that belong either to $\mathrm{BM}_{\mathrm{gt}}$, or to $\mathrm{BM}_{\mathrm{dt}}$, or to both of them.

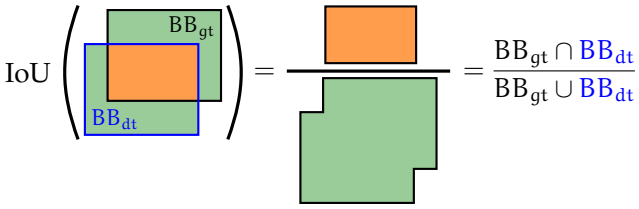


Figure 2.9: Schematic formula of the Intersection over Union (IoU)

2.3.2 Standard Evaluation Metrics

For the evaluation metrics it is important to remember from previous sections that object detection systems output bounding box coordinates (localization) and classifies them (classification) while also producing a score (or confidence score). The same applies to instance segmentation systems with the sole difference that an additional segmentation branch also produces binary masks for localized objects. Apart from the IoU computation, the evaluation metrics presented in this section are the same for object detection and instance segmentation.

Probably the most widely used performance metric for modern object detectors and instance segmentation systems is the Precision vs. Recall curve, and quantitatively, the Average Precision (AP) derived from this curve. The COCO dataset, for instance, on which the performance of virtually all new systems is assessed, incorporates the AP in its challenge (computed under strict requirements regarding the IoU threshold) [38].

Another metric often used in research papers for object detection systems in the field of Autonomous Driving is the Miss Rate (MR) vs. False Positives per Image (FPPI) metric from which the Log Average Miss Rate (LAMR) can be derived as numerical performance assessment [39, 28, 30]. As its name implies, it provides the possibility to set an upper limit of the acceptable false positives per image, which is obviously essential for safety-critical applications such as self-driving cars [30]. Both of these metrics, however, are computed in a very similar way. First, ground truth bounding boxes and detection bounding boxes are matched for each image and category individually. Then, the results are accumulated in order to visualize them with the respective curve. For the sake of simplicity, the process will be explained for object detection. Again, only the IoU computation differs between the metrics for object detection and instance segmentation, so the process relates to instance segmentation as well.

2.3.2.1 Per image and category evaluation

This section explains the evaluation of one image and one category, which must be performed on each image and category individually. Suppose that an object detector performed detection on the entire test set and for one particular image and category, all detection bounding boxes (of size 1×4) and associated scores are

stored in d rows of the arrays $\mathbf{BB}_{dt} \in \mathbb{R}^{d \times 4}$ and $\mathbf{S} \in \mathbb{R}^{d \times 1}$, respectively, where d is the number of detection bounding boxes. Suppose also that the corresponding ground truth bounding boxes for this image and the category are stored in the array $\mathbf{BB}_{gt} \in \mathbb{R}^{g \times 4}$, with g denoting the number of ground truth bounding boxes. Optionally, ground truth labels $\mathbf{L} \in \mathbb{R}^{d}$ with $L_j \in \{0, 1\}$ ($1 =$ ignore and $0 =$ don't ignore) can be assigned to ground truth bounding boxes in order to evaluate only a subset of the test set (i.e. only those bounding boxes with $L_j = 0$). Note, the only differences between the evaluation of different images and categories are the number of ground truth bounding boxes g and detection bounding boxes d.

Matching detection bounding boxes $(BB_{dt})_k$ in $\mathbf{BB}_{dt}$ and ground truth bounding boxes $(BB_{gt})_j$ in $\mathbf{BB}_{gt}$ is performed "greedily", which means that each detection $(BB_{dt})_k$ can match only one ground truth bounding box $(BB_{gt})_j$ and the detections with highest confidence score are matched first [30]. In other words, if one $(BB_{gt})_j$ matches multiple $(BB_{dt})_k$, only the match with the highest scored $(BB_{dt})_k$ counts. Conversely, if one $(BB_{dt})_k$ matches multiple $(BB_{gt})_j$, only the match with highest IoU counts. A proper match presupposes that a bounding box pair with any $(BB_{dt})_k$ and $(BB_{gt})_j$ exceeds a pre-defined IoU threshold IoU_t. Algorithm 2.4 shows how to perform greedy matching. The pseudo code is derived from the COCO API and only slightly changed to adapt the algorithm to the aforementioned arrays [38].

First, the detection bounding boxes in $\mathbf{BB}_{dt}$ and associated scores in $\mathbf{S}$ are sorted in descending order by the scores. Moreover, if only a subset of the dataset is to be evaluated, the ground truth bounding boxes in $\mathbf{BB}_{gt}$ and its corresponding ignore labels in $\mathbf{L}$ are sorted, so that all $(BB_{gt})_j$ with $L_j = 1$ are placed last. Then, for each possible bounding box pair, the IoU is computed and stored in the array $\mathbf{IoUs} \in \mathbb{R}^{d \times g}$ with $(IoUs)_{k,j}$ denoting the IoU of $(BB_{dt})_k$ and $(BB_{gt})_j$.

Final results of the greedy matching function in algorithm 2.4 are stored in the vectors $\mathbf{GTM}$, $\mathbf{DTM}$, and $\mathbf{DTI}$. The elements in $\mathbf{DTM}$ correspond to the elements in $\mathbf{BB}_{dt}$ and are assigned to the index of the ground truth bounding box in $\mathbf{BB}_{gt}$ that they match. Conversely, elements in $\mathbf{GTM}$ correspond to the elements in $\mathbf{BB}_{gt}$ and are assigned to the index of the detection bounding boxes in $\mathbf{BB}_{dt}$ that they match. The indexes in both $\mathbf{DTM}$ and $\mathbf{GTM}$ are 0 if there is no proper match for the respective $(BB_{gt})_j$ or $(BB_{dt})_k$. Finally, the vector $\mathbf{DTI}$ corresponds to $\mathbf{BB}_{dt}$ and its elements are either 1 or 0 based on whether or not the respective $(BB_{dt})_k$ is to be ignored, which is determined by whether or not it matches a (non-regular) ground truth bounding box (whose ignore label is 1). Figure 2.10 visualizes the result vectors along with the sorted arrays and the following paragraphs explain the process in algorithm 2.4 with which the vectors are populated.

An outer for loop iterates over indexes of all d detection bounding boxes

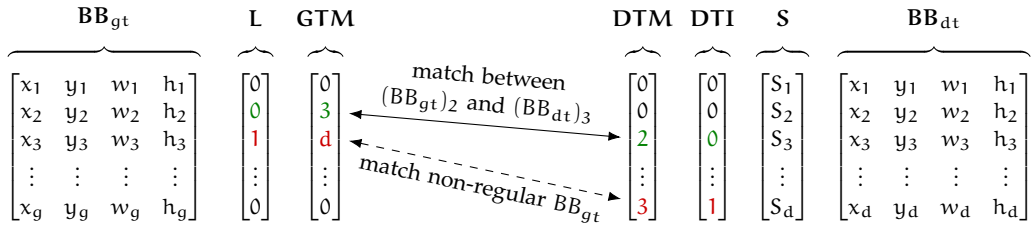


Figure 2.10: Results of the "greedy" matching algorithm. Elements in **GTM** and **DTM** link matching bounding box pairs. If a detection bounding box $(BB_{dt})_k$ matches a non-regular ground truth bounding box $(BB_{gt})_j$ (i.e. with ignore label $L_j = 1$), it is assigned a label $(DTI)_k = 1$ and is, just like the corresponding ground truth bounding box, ignored in further evaluations (red). Conversely, if a detection bounding box matches a regular $(BB_{gt})_j$ (i.e. with ignore label $L_j = 0$), the match counts and flows into subsequent evaluations (green).

$(BB_{dt})_k$ in $\mathbf{BB}_{dt}$ (due to the pre-sorting in descending order from detections with highest score to that with lowest score) and an inner for loop iterates over indexes of all g ground truth bounding boxes $(BB_{gt})_j$ in $\mathbf{BB}_{gt}$ (where $(BB_{gt})_j$ with ignore label $L_j = 1$ are sorted last). In each iteration of the outer loop, the IoU threshold is assigned to the variable IoU and the variable m is initialized with -1 (indicating no match). The inner loop tries to find the best match for the current $(BB_{dt})_k$ in the outer loop and sets the value of m to the index j of $(BB_{gt})_j$ with which the detection matches best in terms of the IoU. The value of m remains equal to -1 if none of the ground truth bounding boxes in the image matches the current detection $(BB_{dt})_k$ properly with respect to the IoU threshold IoU_t and the aforementioned requirement that a ground truth bounding box can only match with one detection bounding box. This is achieved with three if statements explained below.

The first if statement (if $GTM_j > 0$) checks whether the current ground truth bounding box $(BB_{gt})_j$ already matched properly with a previous (higher scored) $(BB_{dt})_k$. As mentioned above, this is the case if the j-th element in **GTM** representing $(BB_{gt})_j$ is already assigned to the index of any of the previous $(BB_{dt})_k$ and thus greater than 0. If this is true, the inner loop proceeds with the next iteration. The second if statement (if $m > -1$ and $L_m = 0$ and $L_j = 1$) checks whether the best match for the current detection is already found. This is the case if m is already assigned to an index of a previous ground truth bounding box (if $m > -1$) whose ignore flag is 0 (if $L_m = 0$) and the current ground truth bounding box $(BB_{gt})_j$ as well as all remaining ground truth bounding boxes (due to the pre-sorting) are to be ignored (if $L_j = 1$). If this is the case, the inner loop can be left to store the match accordingly. If both if statements are false, however, the third if statement (if $(IoUs)_{k,j} < IoU$) checks whether the IoU of the current bounding box pair (stored at index k, j in $\mathbf{IoUs}$) is smaller than the value of IoU,

Algorithm 2.4 `matching` ("Greedy" matching for one image and category) [38]

Input: $\mathbf{BB}_{dt} \in \mathbb{R}^{d \times 4}$ ▷ d detection bbox coordinates $(BB_{dt})_k$ of size 1×4
Input: $\mathbf{S} \in \mathbb{R}^{d \times 1}$ ▷ d scores $S_k \in [0, 1]$ for each $(BB_{dt})_k$
Input: $\mathbf{BB}_{gt} \in \mathbb{R}^{g \times 4}$ ▷ g ground truth bbox coordinates $(BB_{gt})_j$ of size 1×4
Input: $\mathbf{L} \in \mathbb{R}^{g \times 1}$ ▷ g ignore labels $L_j \in \{0, 1\}$ for each $(BB_{gt})_j$
Input: IoU_t ▷ Intersection over Union threshold chosen for evaluation
Output: $\mathbf{DTI} \in \mathbb{R}^{d \times 1}, \mathbf{DTM} \in \mathbb{R}^{d \times 1}, \mathbf{GTM} \in \mathbb{R}^{g \times 1}, \mathbf{BB}_{dt} \in \mathbb{R}^{d \times 4}$, $\mathbf{BB}_{gt} \in \mathbb{R}^{g \times 4}, \mathbf{S} \in \mathbb{R}^{d \times 1}, \mathbf{L} \in \mathbb{R}^{g \times 1}$

function MATCHING($\mathbf{BB}_{dt} \in \mathbb{R}^{d \times 4}, \mathbf{S} \in \mathbb{R}^{d \times 1}, \mathbf{BB}_{gt} \in \mathbb{R}^{g \times 4}, \mathbf{L} \in \mathbb{R}^{g \times 1}, IoU_t$)
▷ sort rows in $\mathbf{BB}_{dt}$ and $\mathbf{S}$ in descending order by scores in $\mathbf{S}$
▷ sort rows in $\mathbf{BB}_{gt}$ and $\mathbf{L}$ so that $(BB_{gt})_j$ and L_j w/ $L_j = 1$ are placed last
$\mathbf{GTM} \leftarrow 0_{g \times 1}$ ▷ initialize zero vector for future results
$\mathbf{DTM} \leftarrow 0_{d \times 1}$ ▷ initialize zero vector for future results
$\mathbf{DTI} \leftarrow 0_{d \times 1}$ ▷ initialize zero vector for future results
for $k = 1, ..., d$ **do** ▷ number of iterations equals number of dt bboxes
$IoU \leftarrow IoU_t$ ▷ initialize IoU with threshold IoU_t for evaluation
$m \leftarrow -1$ ▷ value of m refers to match ($m > -1$) or no match ($m = -1$)
for $j = 1, ..., g$ **do** ▷ number of iterations equals number of gt bboxes
if $GTM_j > 0$ **then**
continue ▷ skip to next gt bbox as this gt bbox already matched
end if
if $m > -1$ and $L_m = 0$ and $L_j = 1$ **then**
break ▷ stop, best match with current dt bbox was already found
end if
if $(IoUs)_{k,j} < IoU$ **then**
continue ▷ skip to next gt bbox unless better match found
end if
$IoU \leftarrow (IoUs)_{k,j}$ ▷ a new best match was found, so store its IoU
$m \leftarrow j$ ▷ store index of gt bbox that matches current dt bbox
end for
if $m = -1$ **then**
continue ▷ skip to next dt bbox, this one doesn't match any gt bbox
end if▷ if this point is reached, a match for the current dt bbox is found
$DTI_k \leftarrow L_m$ ▷ link ignore label of gt bbox to matching dt bbox
$DTM_k \leftarrow m$ ▷ link index of gt bbox to matching dt bbox
$GTM_m \leftarrow k$ ▷ link index of dt bbox to matching gt bbox
end for
return $(\mathbf{DTI}, \mathbf{DTM}, \mathbf{GTM}, \mathbf{BB}_{dt}, \mathbf{BB}_{gt}, \mathbf{S}, \mathbf{L})$
end function

which prior to the inner loop is assigned to the IoU threshold. If this is true, the inner loop proceeds with the next iteration. If this false, however, a potential best match is found and the index j of the current ground truth bounding box $(BB_{gt})_j$ is assigned to m. Moreover, the IoU of the current bounding box pair is assigned to the variable IoU, so that the last if statement in subsequent iterations of the inner loop may check for better matches of the current detection bounding boxes (in the outer loop) with other ground truth bounding boxes (in remaining iterations of the inner loop).

After the inner loop, m is either equal to -1 if there is no proper match for the current detection bounding box or equal to the index of the ground truth bounding box with which it matches best. Consequently, it can be proceeded with the next iteration of the outer loop (i.e. with the next detection bounding box) if $m = -1$. If m is larger than -1, however, the match must be stored in the aforementioned vectors. The index of the ground truth bounding box m involved in the match is assigned to the k-th element in $\mathbf{DTM}$ (i.e. at the index k that represents the current detection bounding box $(BB_{dt})_k$). Analogously, the index of the current detection bounding box k is assigned to the m-th element in $\mathbf{GTM}$ (i.e. at the index m that represents the ground truth bounding box $(BB_{gt})_m$ involved in the match). Finally, the ignore label L_m of the ground truth bounding box involved in the match is assigned to the k-th element in $\mathbf{DTI}$, because detection bounding boxes that match non-regular ground truth bounding boxes (i.e. with $L_j = 1$) must later be ignored.

Repeating this process with all detection bounding boxes results in three arrays (or vectors) $\mathbf{DTI}$, $\mathbf{DTM}$ and $\mathbf{GTM}$ with information about matches for one category in one image. In summary, the values in $\mathbf{DTM}$ (representing detections) and $\mathbf{GTM}$ (representing ground truths) are equal to 0 for no match and equal to the counterpart's index for a match (see figure 2.10). The values in $\mathbf{DTI}$ are 1 if they represent detections that match a ground truth bounding with ignore label $Lj = 1$. In fact, in addition to the overall evaluation, the COCO API also evaluates objects of a specific area ranges, which can be useful for more precise performance evaluations [38]. To this end, detection and ground truth bounding boxes of sizes outside these area ranges may additionally be ignored and thus assigned a 1 in $\mathbf{DTI}$ and $\mathbf{L}$, respectively.

The vectors $\mathbf{DTI}$, $\mathbf{DTM}$ and $\mathbf{GTM}$ are returned along with the sorted arrays $\mathbf{BB}_{dt}$, $\mathbf{BB}_{gt}$, $\mathbf{S}$ and $\mathbf{L}$, which provide corresponding information about the ground truth bounding box coordinates and ground truth ignore labels as well as the detection bounding box coordinates and scores. In fact, subsequent computations for the Precision vs. Recall or MR vs. FPPI require not all data returned from the function in algorithm 2.4. The sorted bounding box coordinates in $\mathbf{BB}_{dt}$ and $\mathbf{BB}_{gt}$, for instance, are redundant in this context. However, the data is required for further evaluations in this work and thus returned accordingly.

For the sake of completeness, it should be noted that the only differences between the per-image and per-category evaluation for object detection and instance segmentation are the arrays of ground truth and detection instances and the way in which they are used to compute the IoUs. The evaluation for object detection sketched here calculates the IoUs with the arrays $\mathbf{BB}_{dt} \in \mathbb{R}^{d\times 4}$ and $\mathbf{BB}_{gt} \in \mathbb{R}^{g\times 4}$ for g ground truth bounding boxes (of size 1×4) and d detection bounding boxes (of size 1×4) with equation 25. The evaluation for instance segmentation, on the other hand, requires arrays $\mathbf{BM}_{dt} \in \mathbb{R}^{d\times h\times w}$ and $\mathbf{BM}_{gt} \in \mathbb{R}^{g\times h\times w}$ for g ground truth masks and d detection masks (of size $h \times w$) to calculate the IoUs with equation 26. These binary masks $(BM_{dt})_k$ and $(BM_{gt})_j$ in the arrays $\mathbf{BM}_{dt}$ and $\mathbf{BM}_{gt}$, respectively, are equal to 1 for pixels that belong to the respective GT or DT object and 0 for the remaining pixels. The entire evaluation process in algorithm 2.4 apart from the IoU computation is for instance segmentation exactly the same.

2.3.2.2 Accumulation of evaluation results

The Precision vs. Recall curve and the MR vs FPPI curve can be computed by accumulating the arrays **DTM**, **L**, **DTI** and **S** for all images and categories. Before explaining the accumulation process, however, it is necessary to first introduce the terms that are to be plotted against each other. They are defined based on the number of TPs, FPs, and False Negatives (FNs). Unmatched detection bounding boxes (i.e. BB_{dt} that doesn't match a BB_{gt} with respect to the "greedy" matching function in algorithm 2.4) count as FPs and unmatched BB_{gt} count as FNs. Detection bounding boxes that match a regular ground truth bounding box (i.e. a BB_{gt} whose ignore label is not 1) count as TPs. This being said, the Precision and Recall are defined as

$$\text{Recall} = \frac{\sum_n^N TP_n}{\sum_n^N TP_n + \sum_m^M FN_m} = \frac{\sum_n^N TP_n}{\sum_k^K P_k}, \tag{27}$$

$$\text{Precision} = \frac{\sum_n^N TP_n}{\sum_n^N TP_n + \sum_j^J FP_j}, \tag{28}$$

where $\sum_k^K P_k = \sum_n^N TP_n + \sum_m^M FN_m$ is the number of Positives (Ps) (i.e. the number of regular ground truth bounding boxes) for the given category. Simply put, the Recall divides the number of correctly predicted examples by the number of examples that are actually positive, whereas the Precision penalizes FPs (i.e. detections that doesn't match any ground truth bounding box). Finally, FPPI is,

as its name implies, the FP rate per image, whereas MR is defined as

$$\begin{aligned} MR &= \frac{\sum_m^M FN_m}{\sum_n^N TP_n + \sum_m^M FN_m} = \frac{\sum_m^M FN_m}{\sum_k^K P_k} = \frac{\sum_k^K P_k - \sum_n^N TP_n}{\sum_k^K P_k} \\ &= 1 - \frac{\sum_n^N TP_n}{\sum_k^K P_k} = 1 - Recall\,, \end{aligned} \tag{29}$$

and can thus be derived from the Recall. In order to plot Precision vs. Recall or MR vs. FPPI, the respective values must be computed for decreasing confidence thresholds. More specifically, the confidence threshold must be varied from high to low and the values of the Precision and Recall or the MR and FPPI must be computed for each decreasing confidence threshold in order to plot them against each other. This can be done straightforwardly by accumulating the arrays (or vectors) **DTM**, **L**, **DTI** and **S** that are returned for each image and category. In this respect, it is important to remember from section 2.3.2.1 that **DTM** represents detection bounding boxes with values equal to or larger than 0 based on whether or not the respective detection matches a ground truth bounding box (see figure 2.10). Moreover, **L** and **DTI** represent ignore labels (1 = ignore, 0 = don't ignore) for ground truth and detection bounding boxes, respectively. Finally, the values in **S** are scores of the detections. For the sake of simplicity, only one category is considered in the following explanation.

First, all arrays must be concatenated, so from here onwards, the arrays **DTM**, **DTI**, **S** and **L** refer to vectors that contain not only data for one image but for all images in the dataset. Next, the former three vectors must be sorted in descending order based on the scores in **S**. Then, based on the concatenated vectors **DTM** and **DTI**, each detection is classified in a TP or FP. The k-th detection in **DTM** is a TP if it's ignore label DTI_k is 0 and the value of DTM_k is greater than 0, as this indicates that the detection bounding box is not to be ignored and matches a ground truth bounding box. Instead, the k-th detection is a FP if both it's ignore label DTI_k and the value DTM_k are equal to 0, since this indicates that it is not to be ignored and doesn't match any ground truth bounding box. Two binary vectors **TP** and **FP** are derived from **DTM** and **DTI** accordingly. Note, the elements of each vector are ordered by descending scores. Hence, the components of the vector **S** are equivalent to the confidence thresholds whereby the corresponding indexes determine, which values in the logical vectors **TP** and **FP** are beyond the respective thresholds. More precisely, the number of values that are true in both logical vectors up to a specific index determine the number of TPs and FPs, respectively, and the value at the corresponding position in vector **S** defines the confidence threshold belonging to those numbers. Thus, simply computing the cumulative sum of the elements of both logical vectors rearrange them accordingly, so that their elements describe the number of TPs and FPs,

respectively, up to the threshold represented by the value at the corresponding index in the vector $\mathbf{S}$.

Based on equation 27, the Recall vector $\mathbf{Rc}$ for category for decreasing confidence thresholds in $\mathbf{S}$ can now be computed by dividing the vector $\mathbf{TP}$ element-wise by the total number of ground truth bounding boxes P, which in turn is the number of ground truth ignore labels in $\mathbf{L}$ that are equal to 0 (i.e. all ground truth bounding boxes that are not to be ignored). Moreover, equation 28 can be applied element-wise with the vectors $\mathbf{TP}$ and $\mathbf{FP}$ in order to compute the corresponding Precision vector $\mathbf{Pr}$ for decreasing confidence thresholds. Plotting $\mathbf{Pr}$ against $\mathbf{Rc}$ results in the Precision vs. Recall curve, in which the ideal performance is put in the upper right corner of the chart (i.e. at Precision = Recall = 1). The area under the curve is called AP and thus ranges from 0 (worst performance) to 1 (ideal performance). If the dataset contains multiple categories, the AP for each category may be averaged in order to derive a final performance assessment of the detection system on all categories. This is referred to as Mean Average Precision (MAP).

The MR vs. FPPI curve can also be derived from the vectors $\mathbf{TP}$ and $\mathbf{FP}$. However, according to equation 29, the MR is also a function of the Recall, so the MR vector $\mathbf{Mr}$ for decreasing confidence thresholds in $\mathbf{S}$ can be computed by simply subtracting the elements in vector $\mathbf{Rc}$ from 1. The corresponding FPPI vector $\mathbf{fppi}$ is straightforwardly the element-wise division of the vector $\mathbf{FP}$ by the total number of images in the dataset. Finally, the vector $\mathbf{Mr}$ can be plotted against $\mathbf{fppi}$, which is usually done with a semi-logarithmic scale. In contrast to the Precision vs. Recall curve, the ideal performance in the MR vs. FPPI curve is put in the lower left corner of the chart. In order to summarize the detector performance in the MR vs. FPPI curve, the MR at 9 evenly spaced data points in log space in the range 10^{-2} to 10^{0} (i.e. at $\mathrm{FPPI} = 10^{-2+\frac{x}{4}}, x \in \{0, ..., 8\}$) are averaged by

$$\mathrm{LAMR} = e^{\left(\frac{1}{9}\sum_{x=0}^{8}\left(\ln\left(MR\left(\mathrm{FPPI}_x = 10^{-2+\frac{x}{4}}\right)\right)\right)\right)}, \tag{30}$$

which is conceptually similar to the AP in the sense that it represents the curve with a single value [30].

In summary, both the Precision vs. Recall and the MR vs. FPPI curve visualize the performance of an object detection or instance segmentation system on a given dataset for the whole range of confidence thresholds. Both curves can thus be used to set a proper confidence threshold for deployment. For instance, if the respective task requires to detect (or segment) most of the instances properly while the number of FPs is not important, the threshold may be set very low to maximize the Recall (and minimize the MR). Conversely, if the number of FPs must be minimized, which is obviously the case for safety-critical applications

such as self-driving cars, the confidence threshold must be increased, which in turn decreases the number of FPPIs and increases the Precision, though at the cost of an increased MR and a decreased Recall. In fact, choosing the confidence threshold is always a tradeoff. As mentioned before, since the MR vs. FPPI curve provides the possibility to set an upper limit of FPPI, it is often used in journal articles in the field of Autonomous Driving.

2.4 Realistic defocus simulation

The link between Computer Vision and Autonomous Driving is already provided with an introduction to Computer Vision in section 2.1 and the presentation of some high-quality Autonomous Driving datasets in section 2.2. Finally, this section links Computer Vision in the field of Autonomous Driving to optics. This is important, given that self-driving cars are highly safety-critical. As shown in section 2.1, Computer Vision systems try to reconstruct properties in images, so the performance is obviously highly dependent on the quality of their inputs. Therefore, they rely on optics or, more specifically, on the output of cameras and thus the quality of the underlying optical systems. These systems, however, suffer unavoidably from aberrations such as coma and astigmatism, leading to blurred images that may influence the performance of object detection or instance segmentation systems. Indeed, today it is possible to construct high-quality spherical lenses whose imaging quality virtually only depends on diffraction effects [40]. This doesn't apply to mass production of lenses, though. Producing lenses on a large scale (e.g. for potential self-driving cars) leads to production tolerances on their aberrations, as individual adjustments for maximizing the performance may not be feasible due to time and cost constraints [41].

The aberrations of optical systems are defined as the departure from the ideal conditions in gaussian optics [40]. They can be described in terms of wave (or wavefront) aberrations, which are the deviations of an actual (aberrated) wavefront from the ideal spherical wavefront [40]. Here, the ideal spherical wavefront describes light refracted through an optical system that results at the position of the systems' exit pupil in a spherical wave with the (gaussian) focus point at its center [40]. Ideal spherical wavefronts that result in this kind of ideal focus points are impossible, though. Instead, real lenses form *spreaded* point sources with various geometrical patterns [40].

Using linear system theory, the effects of an imaging system on a captured scenes with respect to these point sources can be predicted. More specifically, with the point spread function (PSF), the response of an imaging system to an ideal point source (observed on the image plane) can be estimated [3]. As its name implies, the PSF basically predicts the degree of spreading at the respective point on the image plane. Since this is an indicator of aberrations, it is commonly

used to assess the quality of optical systems [42].

The question is now how to simulate these aberrations. To this end, commercial softwares such as OpticStudio by Zemax provide the possibility to simulate real lenses. These lens models allow the parametrization of Zernike-Coefficients, which in turn describe the wavefront profile [43, 42]. Each Zernike coefficient represents a different type of aberration [43]. Therefore, they can be used to introduce a wavefront error [42]. Finally, with the wavefront error a spatially variant PSF can be constructed by Fourier transformation [42].

3 The Spatial Recall Index

The most common evaluation methodologies for performance assessments of object detection and instance segmentation systems are presented in section 2.3. In summary, these evaluation metrics run through all detections from highest to lowest score, classify them into TPs and FPs based on the "greedy" matching algorithm (presented in algorithm 2.4) and finally compute Precision vs. Recall or MR vs. FPPI values for each decreasing confidence threshold using the numbers of TPs, FPs and (indirectly) FNs above the respective thresholds. Indeed, these methodologies provide a clear presentation of the system's performance for the whole range of confidence thresholds. Additionally, they provide a numerical performance assessment by summarizing the curves with the AP and LAMR, respectively.

What these evaluation metrics lack, however, is an assessment of the object detection or instance segmentation performance in dependency of the spatial positions in the field of view, meaning that they don't take into account where in the images TPs, FPs and FNs occur. Again, these metrics just count the number of TPs, FPs and (indirectly) FNs for varying confidence thresholds in order to compute and plot the respective curves (i.e. Precision vs. Recall or the MR vs. FPPI) as shown in section 2.3.2. The optics of camera systems, however, are always spatially variable over the field of view. Therefore, the performance of Computer Vision systems may vary based on the degree of aberration at the spatial positions in the images. To this end, this section introduces a novel metric, called SRI, which does take the spatial dependency of the performance into account in order to evaluate (potential) performance differences of object detection and instance segmentation systems at different positions in the input images.

The SRI basically supplements the Recall value introduced in section 2.3.2 with a spatial domain. As shown in equation 27, the Recall is the ratio of all correctly detected (or segmented) objects and all actually positive objects in a given dataset for a pre-set confidence threshold (i.e. $\sum_n^N TP_n / \sum_k^K P_k$). By computing the ratio of the number of TPs and the number of Ps in a pixel-wise manner, the SRI provides a clear representation of the object detection or instance segmentation performance in dependency on spatial positions in the input images. Section 3.1 and 3.2 explain the SRI for object detection and instance segmentation, respectively.

3.1 Spatial Recall Index for Object Detection

The SRI presupposes that a given dataset contains images with the same height h and width w in order to be able to assign a Recall value to each pixel x, y in the dimension $h \times w$. For a statistically reliable SRI computation, the number of objects and their spatial distribution in the images are important aspects. The ideal case is a dataset with a considerable amount of equally distributed objects whose sizes don't vary much, so that the number of objects and their dimensions are roughly the same at each spatial position in the images. The statistical aspects, however, are described in later sections. Here, the focus is on the metric itself.

Analogous to the common evaluation methodologies described in section 2.3.2, the first step in the SRI computation is to run the "greedy" matching algorithm on each image and category of the dataset and accumulate the results for all images afterwards. Rather than simply assigning binary TP and FP labels to each detection as it is done in section 2.3.2.2, however, the accumulation process for the SRI requires to extract coordinates of ground truth bounding boxes and detection bounding boxes for the computation of Recall values with for spatial positions in the images.

The SRI algorithm is implemented based on the results of the per-image and per-category evaluation of the COCO API and can thus be explained with the arrays returned by the matching function in algorithm 2.4 [38]. Important in this context are the arrays $\mathbf{GTM}$, $\mathbf{L}$, $\mathbf{S}$, $\mathbf{BB}_{dt}$ and $\mathbf{BB}_{gt}$. As shown in figure 2.10, the array $\mathbf{GTM}$ represents all ground truth bounding boxes in $\mathbf{BB}_{gt}$ and its elements indicate with positive values from which detection bounding box in $\mathbf{BB}_{dt}$ they are matched, and with 0 that they aren't matched by any detection bounding box. Furthermore, the labels in $\mathbf{L}$ determine, which ground truth bounding boxes are to be considered in the evaluation, and the array $\mathbf{S}$ contains the scores of detection bounding boxes in $\mathbf{BB}_{dt}$. These per-image and per-category arrays are accumulated to produce the arrays described in the following paragraph. For reason of simplicity, only one category is considered.

The array $\mathbf{GT} \in \mathbb{R}^{K \times 4}$ contains all K regular ground truth bounding boxes in the dataset. The array $\mathbf{GTM} \in \mathbb{R}^{K \times 1}$ (now referring to a vector for all K regular ground truth bounding boxes in the dataset) contains corresponding indexes to access the counterpart's TPs if the respective ground truth bounding boxes are matched, while its values are 0 for not detected ground truth bounding boxes. The TPs may be accessed from a third array, $\mathbf{TP} \in \mathbb{R}^{M \times 4}$, where M denotes the number of TPs. Finally, the last array $\mathbf{S} \in \mathbb{R}^{M \times 1}$ contains scores of the detection bounding boxes in $\mathbf{TP}$. These arrays along with a confidence threshold and the dimension $h \times w$ of images in the dataset are passed to the SRI function presented in algorithm 3.1. How the function computes the SRI is explained in the following paragraph.

First, three arrays with the dimension $h \times w$ (equal to the size of images in

Algorithm 3.1 SRI (Spatial Recall Index for one category)

Input: thr_c ▷ pre-set confidence threshold for evaluation
Input: h, w ▷ height h and width w of images in the dataset
Input: $\mathbf{GT} \in \mathbb{R}^{K\times 4}$ ▷ K ground truth bbox coordinates GT_k of size 1×4
Input: $\mathbf{TP} \in \mathbb{R}^{M\times 4}$ ▷ M true positive bbox coordinates TP_m of size 1×4
Input: $\mathbf{S} \in \mathbb{R}^{M\times 1}$ ▷ M scores S_m of TPs in **TP**
Input: $\mathbf{GTM} \in \mathbb{R}^{K\times 1}$ ▷ K indexes GTM_k of TPs in **TP** that match ground truth bboxes GT_k in **GT**, while the elements GTM_k are 0 for unmatched GT_k
Output: $SRI \in \mathbb{R}^{h\times w}$ ▷ Spatial Recall Index for images of size $h \times w$

function SRI($\mathbf{GT} \in \mathbb{R}^{K\times 4}, \mathbf{TP} \in \mathbb{R}^{M\times 4}, \mathbf{GTM} \in \mathbb{R}^{K\times 1}, \mathbf{S} \in \mathbb{R}^{M\times 1}, thr_c, h, w$)
  $GTD \leftarrow 0_{h\times w}$ ▷ initialization of array for ground truth distribution
  $TPD \leftarrow 0_{h\times w}$ ▷ initialization of array for true positive distribution
  $SRI \leftarrow 0_{h\times w}$ ▷ initialization of array for SRI
  for $k = 1, ..., K$ **do** ▷ number of iterations equals number of gt bboxes
    $GTD(x, y) \leftarrow GTD(x, y) + 1 \; \forall \; (x, y) \in GT_k$ ▷ increment pixels of GT_k
    if $GTM_k > 0$ **then** ▷ check for a matching TP
      $n \leftarrow GTM_k$ ▷ k-th element is index of TP in **TP** that matches GT_k
      if $S_n > thr_c$ **then** ▷ check if score exceeds confidence threshold
        $TPD(x, y) \leftarrow TPD(x, y) + 1 \; \forall \; (x, y) \in [TP_n \cap GT_k]$ ▷ add intersec.
      end if
    end if
  end for
  $SRI_{h\times w}(x, y) \leftarrow TPD_{h\times w}(x, y) \oslash GTD_{h\times w}(x, y) \; \forall \; GTD_{h\times w}(x, y) > 0$
  return (SRI)
end function

the dataset) are initialized with zeros. The arrays GTD and TPD refer to the Ground Truth Distribution and True Positive Distribution, respectively, and the array SRI is the Spatial Recall Index returned by the function. A loop, whose number of iterations equals the number of ground truth bounding boxes GT_k, increments in each iteration the pixels that belong to GT_k by 1 in the Ground Truth Distribution GTD. In other words, the pixels or elements in GTD that are within the respective ground truth bounding box ($x, y \in GT_k$) are incremented by 1. Moreover, in each iteration, an if statement checks whether there is a TP (if $GTM_k > 0$) that matches the ground truth bounding box GT_k. If this is false, the function proceeds with the next iteration and thus the next ground truth bounding box. If this is true, however, the detection's score S_n (indexed by $n = GTM_k$) is accessed from **S** and another if-statement checks whether it exceeds the pre-set confidence threshold (if $S_n > thr_c$). If this is also true, the respective detection TP_n (also indexed by $n = GTM_k$) is accessed from the array **TP** in order to compute the intersection area $TP_n \cap GT_k$. Then, the pixels of this particular area (i.e. all pixels that are within the intersection area) are incremented

by 1 in the TPD (i.e. $\forall\ (x, y) \in [TP_n \cap GT_k]$).

Finally, after the process is carried out for each regular ground truth bounding box GT_k, the array $GTD(x, y)_{h \times w}$ represents for each pixel x, y (within $h \times w$) how often it is part of a ground truth bounding box in the given dataset, whereas the array $TPD(x, y)_{h \times w}$ represents for each pixel x, y how often it is located within an overlap area $TP_n \cap GT_n$. The total number of valid TPs that flow into the computation after suppressing the detections whose scores don't exceed the given confidence threshold may be denoted by N. Finally, an element-wise devision $GTD(x, y)_{h \times w} \oslash TPD(x, y)_{h \times w}$ denoted by the Hadamard division $\oslash$ for pixels x, y that are larger than 0 in the denominator (i.e. pixels that belong to at least one ground truth bounding box GT_k) results in the $SRI(x, y)_{h \times w}$ returned by the function. Here, it is import to highlight again that the True Positive Distribution in the numerator is based on the intersection area of the TPs rather than the entire area of the TP bounding boxes. The purpose of this is to penalize inaccurate detections regardless of the pre-defined IoU threshold used for evaluation.

The SRI for one category on a dataset with images of size $h \times w$ may now be defined in mathematical terms. To this end, all N TPs for one category accumulated from the results of the "greedy" matching algorithm in the aforementioned manner along with the respective Ps that they match are defined as N bounding box pairs

$$\big((GT)_n, (TP)_n\big) \in \left\{ (GT, BB_{dt}) \;\middle|\; \begin{array}{l} IoU\big((GT)_k, (BB_{dt})_j\big) > thr_{IoU}, \\ score_{(BB_{dt})_j} > thr_{confidence} \end{array} \right\}. \quad (31)$$

As explained before in algorithm 2.4, these bounding box pairs are only valid if they exceed a pre-defined IoU threshold thr_{IoU}. Moreover, as shown in algorithm 3.1, the bounding box pairs are limited by considering only detection bounding boxes $(BB_{dt})_j$ whose scores $scr_{(BB_{dt})_j}$ exceed a pre-defined confidence threshold thr_c. As mentioned above, the total number of TPs without a lower limit of acceptable scores, which is taken as input by the function in algorithm 3.1, is denoted as M and the suppression of detections beyond the given threshold (within the function) eventually results in N remaining TPs and thus in N valid bounding box pairs defined by equation 31. With these N bounding box pairs consisting of $TP_n \in 1 \times 4$ and $GT_n \in 1 \times 4$ along with all K (regular) ground truth bounding boxes $GT_k \in 1 \times 4$, the SRI in mathematical terms is defined as

$$SRI(x, y) = \left[\sum_{n=1}^{N} \begin{cases} 1 & (x, y) \in [TP_n \cap GT_n] \\ 0 & else \end{cases} \right] \oslash \left[\sum_{k=1}^{K} \begin{cases} 1 & (x, y) \in GT_k \\ 0 & else \end{cases} \right], \quad (32)$$

with $(x, y) \in [TP_n \cap GT_n]$ denoting all pixels x, y in $h \times w$ that belong to the intersection area of the n-th bounding box pair $\big((GT)_n, (TP)_n\big)$ and $(x, y) \in GT_k$

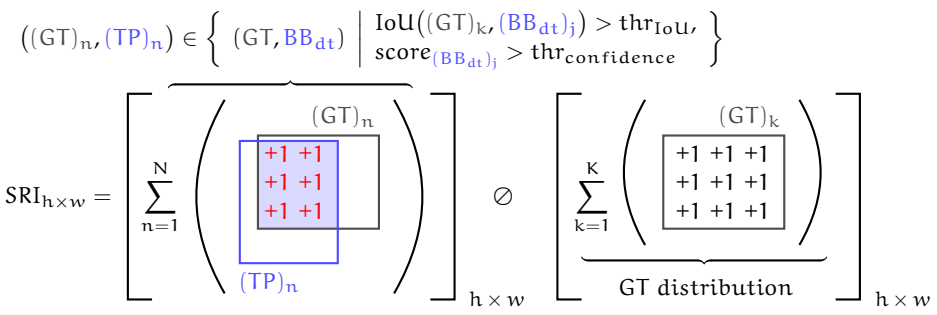


Figure 3.1: Schematic formula of the Spatial Recall Index for Object Detection.

denoting all pixels x, y in $h \times w$ that belong to the k-th ground truth bounding box. The element-wise division denoted by the Hadamard division $\oslash$ divides all pixels in the numerator by the respective pixels in the denominator with the sole exception of those pixels in the denominator whose values are equal to 0, meaning that the element-wise division is performed only with pixels in $h \times w$ that belong to at least one ground truth bounding box GT_k. For the sake of clarity, a schematic formula of the SRI is presented in figure 3.1. It highlights in red that for each one of the N valid bounding box pairs $\left((GT)_n, (TP)_n\right)$ only pixels within the intersection area are incremented by 1 to construct the True Positive Distribution $TPD_{h \times w}$ in the numerator, whereas the Ground Truth Distribution $GTD_{h \times w}$ in the denominator is straightforwardly constructed by incrementing the pixels within the ground truth bounding boxes GT_k by 1 for each one of the K ground truth bounding boxes.

3.2 Spatial Recall Index for Instance Segmentation

The SRI for instance segmentation extends the spatial performance assessment in terms of accuracy compared to the process outlined in section 3.1. Instead of computing pixel-wise Recall values with bounding box coordinates, the SRI is computed based on binary masks. The implementation of the algorithm, however, is very similar to the implementation described in section 3.1.

First, the outputs from the "greedy" matching function in algorithm 2.4 must be accumulated. The matching function is applied to instance segmentation instead of object detection, so the function's outputs and thus the accumulation process differ regarding the instances. Rather than accumulating bounding boxes of the size 1×4, binary masks of the size $h \times w$ need to be accumulated. More specifically, instead of concatenating bounding boxes coordinates to the arrays $\mathbf{GT} \in \mathbb{R}^{K \times 4}$ and $\mathbf{TP} \in \mathbb{R}^{M \times 4}$, binary masks of the size $h \times w$ are concatenated to the arrays $\mathbf{GT} \in \mathbb{R}^{K \times h \times w}$ and $\mathbf{TP} \in \mathbb{R}^{M \times h \times w}$, while K and M still denote the number of Ps and TPs, respectively.

The SRI function for instance segmentation is conceptually similar to the SRI

(a) Ground truth instance (b) True Positive instance

Figure 3.2: Example pair of a (a) ground truth instance and a (b) corresponding True Positive instance located at the left side of an input image. It can be seen that the segmentation (with a fairly high score of 0.9913) in (b) is pretty precise with the exception of some minor inaccuracies such as the segmentation of the right sight mirror. The IoU of this example pair is 0.965.

function for object detection in algorithm 3.1. It loops through all K ground truth masks $GT_k \in \mathbb{R}^{h \times w}$ in $\mathbf{GT} \in \mathbb{R}^{K \times h \times w}$ and adds in each iteration the binary mask to the Ground Truth Distribution $GTD_{h \times w}$. The binary masks have the same dimension as the input images with values equal to 1 for pixels that belong to the segmented object and values equal to 0 for the remaining pixels. Consequently, by adding the binary mask to the GTD, the pixels that belong to the object are incremented by 1. In each iteration the algorithm also checks whether or not there is a TP mask that matches the ground truth mask with a score exceeding the pre-defined confidence threshold. If this is false, the algorithm proceeds with the next iteration. If this true, however, the intersection of the ground truth mask GT_k and the corresponding TP mask TP_n is incremented by 1 in the True Positive Distribution TPD. To this end, a binary mask of size $h \times w$ with values equal to 1 for all pixels that are 1 in both GT_k and TP_n and 0 for all remaining pixels is added to the TPD. Finally, after the loop, the SRI is computed by an element wise division of the TPD and GTD just like in algorithm 3.1.

In order to illustrate the SRI computation for instance segmentation, figure 3.2 shows exemplary a proper pair of a ground truth instance and corresponding TP instance with an IoU of 0.965. The ground truth instance in figure 3.2 a is located at the left side of an input image. The pixels that are equal to 1 in the respective binary mask are colored in green. Analogously, the pixels that are equal to 1 in the binary mask for the corresponding TP instance in figure 3.2 b (outputted by the instance segmentation system) are colored in blue. It can be seen that the segmentation is pretty precise with the exception of some minor inaccuracies such as the segmentation of the right sight mirror.

The example instances in 3.2 a and 3.2 b are used to visualize the SRI com-

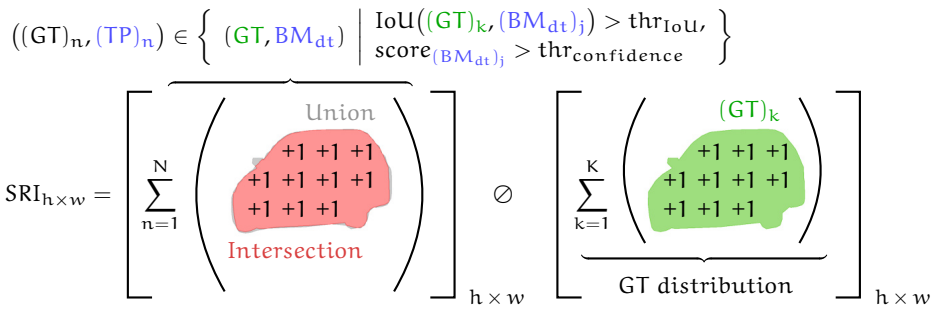


Figure 3.3: Schematic formula of the Spatial Recall Index for Instance Segmentation.

putation schematically in figure 3.3. The numerator shows the union area (gray) overlapped by the intersection area (red) of both instances. In order to increment only pixels within the intersection area, a binary mask of the size $h \times w$ with values equal to 1 in the intersection area and equal to 0 beyond the intersection area is added to the $TPD_{h\times w}$. In the denominator, the binary mask of the ground truth instance in figure 3.2 a is added to the $GTD_{h\times w}$, which obviously increments all pixels that belong to the object. Finally, the process is repeated with all N pairs $\left((GT)_n, (TP)_n\right)$ in the numerator and all K ground truth instances in the denominator before an element wise division of the resulting TPD and GTD is performed to compute the $SRI_{h\times w}$.

In summary, the SRI proposed in this section is a metric for the performance assessment of object detection and instance segmentation systems. It can be used to evaluate the system's performance in dependency of the spatial positions of objects in the input images. To this end, the SRI evaluates the performance of an object detection or instance segmentation system on a dataset with images of the size $h \times w$ by assigning a Recall value to each pixel of a two dimensional matrix with the same size as the dataset's images. Inaccuracies are penalized by taking only intersection areas rather than entire TP areas into account. FPs, on the other hand, are not directly penalized. However, a reasonable confidence threshold for a potential deployment of the object detection or instance segmentation system may be set based on the common evaluation metrics such as the MR vs. FPPI introduced in section 2.3. All detections or instance segmentations with a score beyond the pre-set confidence threshold don't flow into the SRI computation.

4 Evaluating defocus conditions

This section describes the evaluation of Computer Vision systems in the field of Autonomous Driving under simulated effects of defocus. The objectives are to make the defocus conditions physically realistic and to evaluate the impact they may have on the performance of Computer Vision systems as accurately as possible. Since optics of cameras are always spatial variable in the field of view, simulating realistic defocus conditions leads to varying effects of defocus at different spatial positions. Hence, the performance of Computer Vision systems must be assessed by taking the spatial dimension into account. To this end, the performance is evaluated not only with standard evaluation metrics described in section 2.3.2 but also with the SRI introduced in section 3.

First, it must be addressed how to simulate real-world driving scenes in order to evaluate the performance of state-of-the-art Computer Vision systems in the field of Autonomous Driving. This is a challenging task in and of itself as it requires to reproduce all kinds of scenes, weather conditions, surroundings, etc. that a Computer Vision system might encounter. Moreover, the scenes must be labeled in order to be able to evaluate the performance of Computer Vision systems that operate in these scenes. The most feasible option is to rely on professionally developed Autonomous Driving datasets such as those presented in table 2.1. The approach in this work is to select datasets that contain appropriately labeled data and whose specifications fit the statistical requirements of the evaluation. Section 4.1 describes the considerations in selecting the datasets.

Thereafter, section 4.2 presents state-of-the-art Computer Vision systems that are used for evaluation. Object detection and instance segmentation are considered and the focus is on pedestrians and cars. The overall aim here is to provide a solid baseline performance on the selected datasets in these tasks.

Then, section 4.3 shows how images of the selected datasets are manipulated in order to provide degraded datasets with spatially different defocus conditions for the evaluation. Several degraded datasets are produced in order to compare the performance under different effects of defocus with the baseline performance evaluated on the original datasets.

Finally, section 4.4 presents the experimental setup with which the performances of object detection and instance segmentation systems under effects of defocus are evaluated both with standard metrics and the SRI.

4.1 Selection of datasets

With the growing research interest in self-driving cars, Autonomous Driving datasets have received increasing research attention in the recent decade. Various widely used datasets are listed in table 2.1. While many of these datasets such as CityPersons and EuroCity Persons are solely person detection datasets, others like CityScapes and BDD100K contain multiple labeled objects besides persons such as vehicles, traffic signs and traffic lights [33, 34, 32, 36]. This work focusses on pedestrians and cars, so one selection criterium is the availability of labeled data for these two categories. Moreover, object detection and instance segmentation are subjects of this work, so labeled data must come with bounding box coordinates and more complicated annotations for instance segmentation.

Further selection criteria are the number of pedestrians and cars as well as the diversity of scenes. These statistical aspects play an important role in the performance evaluation under effects of defocus, especially with regards to spatial performance assessments. For object detection, the Berkeley Deep Drive dataset (BDD100K) is chosen as it scores high in the two aforementioned criteria. It contains 100k images that are extracted from 100k different video scenes captured in various cities throughout the United States [36]. The dataset contains more than 1M cars and 135,732 persons (pedestrians and riders), which even surpasses the number of persons that are labeled in person detection datasets such as KITTI and CityPersons [36]. Only the average number of persons per image is with approximately 1.2 lower than CityScapes and KITTI because the BDD100k dataset also contains non-city scenes such as highways [36]. However, on the other hand non-city scenes contribute to the scene diversity.

In fact, the diversity is besides the scale a distinct feature of the BDD100K dataset [36]. Each image in the dataset is tagged with three labels specifying the weather, time of the day, and scene type, respectively. The diversity of scenes ranges from city streets over tunnels, residential areas and parking lots to gas stations. Various weather conditions occur during these scenes, which are labeled by "clear", "rainy", "snowy", "cloudy", "foggy" and "overcast". Finally, images are collected in three distinct times of day labeled by "daytime", "night" and "dawn/dusk".

The individual objects are flagged with two boolean labels "occluded" and "truncated", indicating the object's visibility. The former, "occluded", indicates whether or not an object is partly covered by other parts in the image, whereas the latter, "truncated", indicates whether the entire object is within the image or only parts of it. Table 4.1 presents the instance statistics for pedestrians and cars in the BDD100K training and validation set with respect to the times of day and the object's visibilities. The percentage of instances that are occluded and truncated indicate how challenging the dataset is. Roughly 58% of pedestrians and roughly 68% of cars are occluded in the training and validation set. Moreover,

Subset	Time of day	# Images	# Pedestrians (o/t/f)	# Cars (o/t/f)
Train	all	69,863	92,159 (58% / 3% / 40%)	700,703 (68% / 9% / 25%)
Train	daytime	36,728	66,724 (60% / 3% / 38%)	402,222 (71% / 9% / 23%)
Train	night	27,971	19,015 (54% / 3% / 44%)	242,241 (63% / 9% / 29%)
Train	dawn/dusk	5,027	6,352 (54% / 3% / 43%)	55,443 (70% / 9% / 24%)
Train	undefined	137	68 (63% / 4% / 32%)	797 (65% / 13% / 28%)
Val	all	10,000	13,425 (58% / 3% / 39%)	102,837 (69% / 9% / 25%)
Val	daytime	5,258	9,476 (59% / 3% / 39%)	58,283 (71% / 9% / 23%)
Val	night	3,929	2,882 (56% / 3% / 42%)	35,751 (65% / 9% / 28%)
Val	dawn/dusk	778	1,060 (57% / 3% / 40%)	8,649 (69% / 9% / 25%)
Val	undefined	35	7 (57% / 14% / 29%)	154 (59% / 8% / 34%)

Table 4.1: Statistics for pedestrians and cars in the BDD100K train and val set with respect to the times of day and the object's visibility (occluded/truncated/fully visible).

the percentage of pedestrians and cars that are truncated are approximately 3 and 9, respectively. Only about 25% of cars and roughly 40% of pedestrians are neither occluded nor truncated and thus fully visible.

In summary, the large-scale BDD100K dataset with its scene diversity is chosen for object detection as statistical aspects such as the number of instances and the object's spatial distribution in the images play important roles in this work. Labels for instance segmentation, however, were during the practical work on this thesis not yet released by the BDD100K dataset, so the evaluation is extended to the CityScapes datasets. It is captured in various cities in Germany and Switzerland during daytime and provides fine-grained instance-level semantic labels for persons, cars and six additional categories. Table 4.2 shows its instance statistics with regards to pedestrians and cars. Note, just like the BDD100K dataset, CityScapes makes a distinction between pedestrians and riders. The number of pedestrians in table 4.2 only refers to the former.

Subset	# Images	# Pedestrians	# Cars
Train	2,975	17,395	26,180
Val	500	3,278	4,524

Table 4.2: Statistics for pedestrians and cars in the CityScapes train and val set.

4.2 Selection of Computer Vision Algorithms

The overall goal in the selection of Computer Vision algorithms is to provide a solid baseline performance in object detection and instance segmentation on the selected datasets for the respective tasks. Moreover, the selection is based on two criteria, namely (1) that the selected systems are trained on different datasets than the datasets used for evaluation, and that (2) the systems can be applied on the selected datasets without any modifications. The reasons for that are described in the following paragraph.

A statistically reliable evaluation in dependency of spatial positions in the input images by the SRI metric requires large amounts of data. However, datasets usually don't release ground truth labels for test sets due to Computer Vision challenges, so only labeled data for training and validation sets are available. This applies also to the BDD100K and CityScapes datasets. This means according to the explanation in section 2.2 that for a system trained with the training and validation set of, say, the BDD100K dataset, a performance evaluation that is representative of the performance on unseen data with this dataset is due to the lack of annotations in the test set not possible. Finally, a Computer Vision system with a solid baseline performance on a given dataset on which it is not trained ensures that it doesn't overfit the idiosyncrasies of the selected dataset. This, in turn, makes the evaluation under effects of defocus more reliable, because for a Computer Vision system trained on a different dataset, the original dataset (for baseline evaluation) as well as the degraded datasets (for evaluations under effects of defocus) are completely new, unseen data.

Two Computer Vision systems are used for evaluation: HTC and Cascade Mask R-CNN. Their architecture are explained in section 2.1.5. The former, HTC, is used for pedestrian detection on the BDD100K dataset and the latter, Cascade Mask R-CNN, is used for object detection and instance segmentation for the category "car" on BDD100K and CityScapes, respectively.

4.2.1 Hybrid Task Cascade

As shown in table 2.1, pedestrian detection received widespread research attention in recent decades with many Autonomous Driving datasets focussing solely on pedestrians. To provide a solid baseline performance in pedestrian detection, it is thus reasonable to rely on a system trained specifically for the detection of pedestrians. On the Papers with Code website, an open source community project lead by Facebook AI Research, the Pedestron repository is ranked number 1 on two of the pedestrian detection datasets listed in table 2.1: Caltech and CityPersons [44, 45]. It provides multiple detection systems trained on various pedestrian detection datasets and is thus well suited for the evaluation in this work. With the Precision vs. Recall metric explained in section 2.3.2, Pedestron's

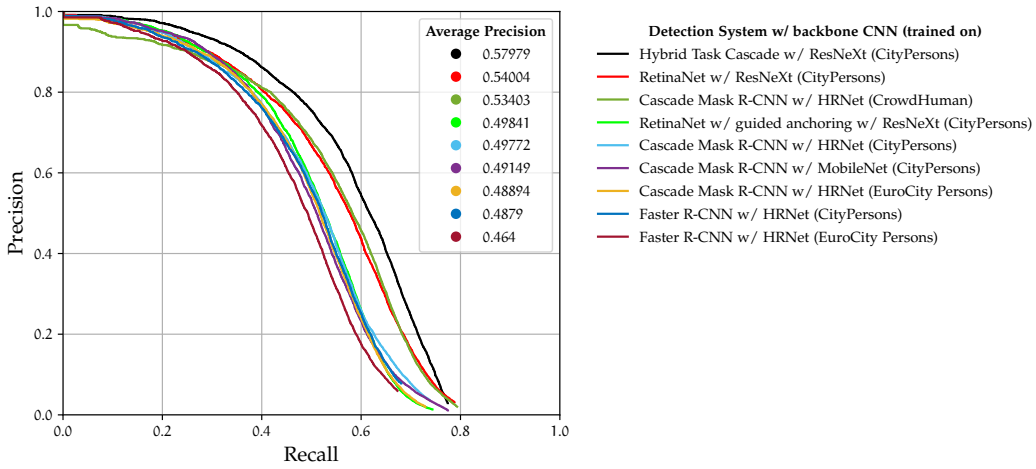


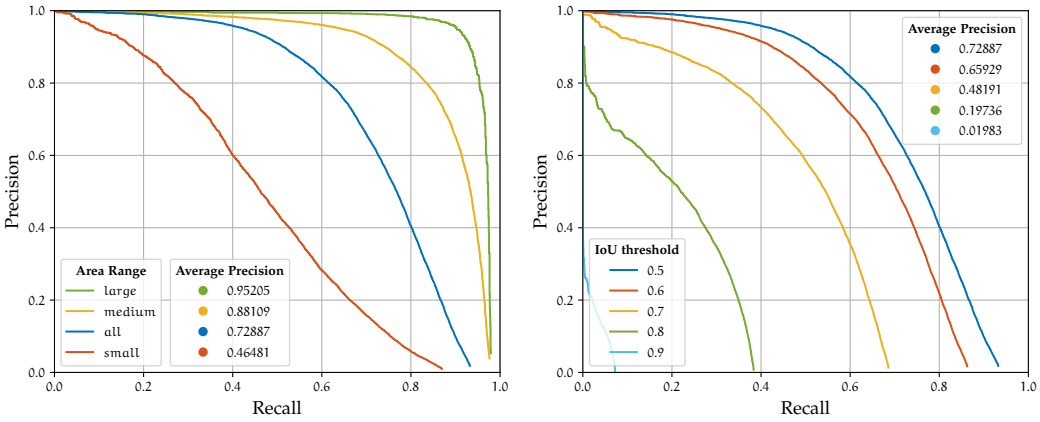
Figure 4.1: Evaluation of systems from the Pedestron repository [44] on the BDD100k validation set. The IoU threshold for the computation of the Precision vs. Recall curves is set to 0.5.

detection systems are evaluated on all 10k images of the BDD100K validation set, which according to table 4.1 consists of 13,425 pedestrians. The IoU threshold for the Precision vs. Recall computation is set to 0.5.

Figure 4.1 shows that HTC performs best on the BDD100K dataset. It is important to note, though, that HTC is trained on CityPersons, a dataset that consists only of images during daytime. Moreover, it labels pedestrians differently than the BDD100K dataset. Instead of drawing bounding boxes around the visible parts of pedestrians, which is the approach in the BDD100K dataset, the CityPersons dataset includes also occluded parts of pedestrians in the scope of bounding boxes. This may lead to FPs for occluded pedestrians in the BDD100K dataset, which by CityPersons' standatds are actually correctly detected.

Therefore, the baseline performance of HTC shown in figure 4.2 is evaluated only on fully visible pedestrians in daytime images of the BDD100K training set by ignoring pedestrians with the flag "occluded" or "truncated". According to table 4.1 fully visible pedestrians account for approximately 40% of pedestrians in the dataset. Figure 4.2 a presents the Precision vs. Recall curves for various bounding box area ranges evaluated with an IoU threshold of 0.5 and figure 4.2 b shows additional Precision vs. Recall curves evaluated for all bounding box area ranges with different IoU thresholds.

With an average precision of 72.9% on all fully visible pedestrians of daytime images in the BDD100K training set computed with an IoU threshold of 0.5, HTC serves as solid benchmark for the evaluation in this work. Figure 4.2 a shows also that the performance for large and medium-sized bounding boxes are very high and that the slope of the curve for the overall performance is mostly due to small bounding boxes.



(a) Area Ranges (IoU threshold = 0.5) (b) IoU thresholds (all area ranges)

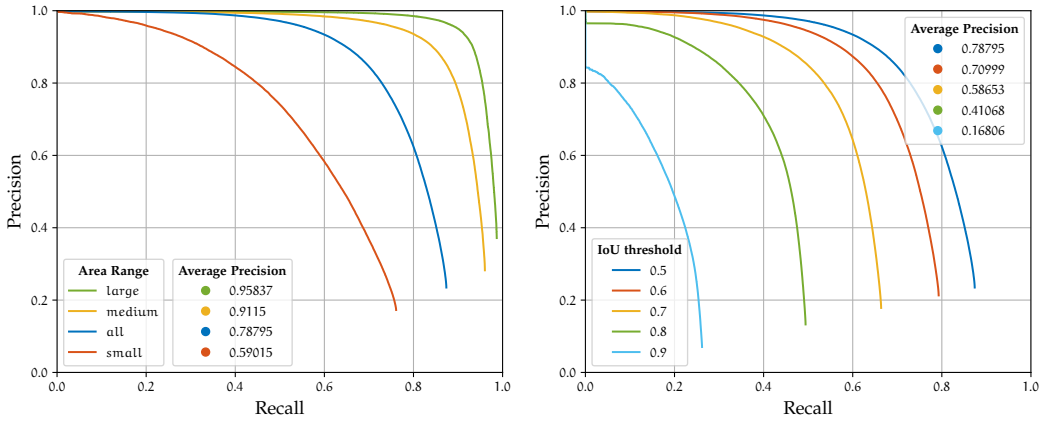
Figure 4.2: Baseline object detection performance of HTC with backbone ResNeXt (trained on CityPersons) evaluated for the category "pedestrian" with the Precision vs. Recall curve on a subset (images with the flag "daytime") of the BDD100k training set. Only fully visible "pedestrian" bounding boxes are considered by ignoring instances with the flag "occluded" or "truncated". Bounding box area ranges in (a) are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

4.2.2 Cascade Mask R-CNN

Similar to pedestrian detection, car detection received widespread research attention. However, none of the most widely used Autonomous Driving datasets focusses solely on cars. Instead, labels for cars in Autonomous Driving datasets are usually among labels for various other objects such as pedestrians, traffic signs and other vehicles, so publicly released Computer Vision are rarely trained solely for car detection (see table 2.1).

In this work, Cascade Mask R-CNN X152 from Facebook's Detectron2 repository is chosen [2, 6]. It is trained on COCO for object detection and instance segmentation and meets all criteria applied before: Firstly, it is neither trained on BDD100K nor on CityScapes and, secondly, it is capable of detecting and segmenting cars [6, 38]. Finally, it generalizes well, which can be seen on the evaluation results for car detection on daytime images of the BDD100K training set in figure 4.3. The baseline evaluation follows that of HTC in figure 4.2 with the sole exception that all objects are considered instead of only fully visible ones, which makes a total of 402,222 car instances in 36,728 images (see table 4.1).

Just like HTC, Cascade Mask R-CNN performs well for medium-sized and large bounding boxes, while the overall performance is mostly impaired by small bounding boxes. Still, the overall performance in the evaluation with an IoU threshold of 0.5 is with 78.7% very high, considering that in daytime images of the BDD100K dataset 71% of cars are occluded and 23% of cars are truncated (see

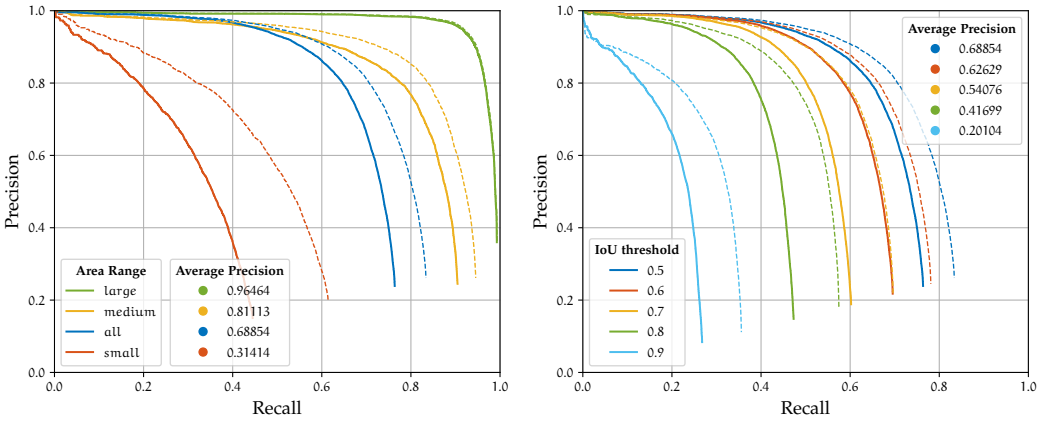


(a) Area Ranges (IoU threshold = 0.5) (b) IoU thresholds (all area ranges)

Figure 4.3: Baseline object detection performance of Cascade Mask R-CNN (trained on COCO) evaluated with the Precision vs. Recall curve on a subset of the BDD100k training set (images with the flag "daytime"). Bounding box area ranges in (a) are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

table 4.1). The average precision for medium-sized and large car bounding boxes computed with an IoU threshold of 0.5 is 91.1% and 95.8%, respectively.

For the sake of completeness, the baseline performance of Cascade Mask R-CNN is also shown for instance segmentation in figure 4.4. The Precision vs. Recall curves are evaluated for the category "car" on the CityScapes training and validation set, which according to table 4.2 consists in total of $2,975 + 500 = 3,475$ images with $26,180 + 4,524 = 30,704$ labeled cars. The overall instance segmentation performance of Cascade Mask R-CNN evaluated with an IoU threshold of 0.5 is with an Average Precision of 68.8% about 10% below the Average Precision computed with the same IoU threshold for object detection before (cf. figure 4.3 and 4.4). To provide additionally a comparison between the object detection and instance segmentation performance on the same data, the dashed lines in figure 4.4 represent the object detection performances of Cascade Mask R-CNN on the CityScapes dataset. Dashed and continuous lines with the same colors indicate that the respective Precision vs. Recall curves for object detection and instance segmentation result from the same evaluation settings in terms of the IoU threshold and area ranges. The object detection performances exceed the corresponding instance segmentation performances in all cases, while the largest performance differences can be observed for small instances in figure 4.4 a.



(a) Area Ranges (IoU threshold = 0.5) (b) IoU thresholds (all area ranges)

Figure 4.4: Baseline instance segmentation performance of Cascade Mask R-CNN X152 (trained on COCO) evaluated for the category "car" with the Precision vs. Recall curve on a the CityScapes training and validation set. As a comparison, the corresponding curves for object detection are plotted with the same colors in dashed lines. Area ranges of instance level semantic labels (and bounding boxes) in (a) are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

4.3 Image Degradation

With the selection of datasets and Computer Vision algorithms in section 4.1 and 4.2, respectively, the ground work for the evaluation in this thesis is done. A solid baseline performance for object detection and instance segmentation on a large number of challenging driving scenes is provided. The main topic, however, is to examine if and how the performance of these Computer Vision systems in real world driving scenes changes under realistic defocus conditions. More specifically, the goal is to evaluate the performance in object detection and instance segmentation under different effects of defocus in comparison with the respective baseline performance. To this end, the chosen datasets are degraded with the use of an optical model that simulates a realistic optical lens and which allows for parameterization to simulate different effects of defocus. On the basis of section 2.4, the optical model is briefly explained in section 4.3.1 and the image degradation is presented under the name "Defocus Study" in section 4.3.2.

4.3.1 Optical Model

With the commercial software OpticStudio by Zemax, a so-called Cooke-Triplet, which is a three-element lens configuration, is simulated. For images of the BDD100K dataset, whose resolution is 1280×720, a setup with f#2.8, focal length 12.5mm, pixel size 4.46µm and diagonal FOV$= \pm 25°$ is applied. For images

of the CityScape dataset, whose resolution is 1024×2048, a setup with f#2.2, focal length 25mm, pixel size $4.84\mu m$ and FOV$= \pm 25°$ is established. Section 4.3.2 describes how these optical models are used to simulate different defocus conditions and degrade images of the BDD100K and the CityScapes accordingly.

4.3.2 Defocus Study

As mentioned in section 2.4, the simulated lens models allow the parametrization of Zernike-Coefficients. These Zernike-Coefficients describe the wavefront profile with every individual coefficient representing a different type of aberration [43]. One of these coefficients is the defocus coefficient Z_2^0 [43]. In order to simulate different defocus conditions, this particular coefficient is parameterized by adding a constant offsets ranging from -1.25 to $+1.25$. More specifically, the offsets Z_Δ are added to the defocus coefficient Z_2^0 in Zernike space by

$$\tilde{Z}_2^0 = Z_2^0 + Z_\Delta\,, \quad Z_\Delta \in \{-1.25, -0.75, -0.5, 0, +0.5, +0.75, +1.25\}\,,$$

where $Z_\Delta = 0$ refers to the nominal position with no offset being added. It is important to note, though, that even without offset, i.e. with $Z_\Delta = 0$, the simulated lens already shows a spatially dependent defocus Z_2^0 due to the field curvature. The offset then adds or cancels the original contribution that the field curvature of the lens (with $\tilde{Z} = Z_2^0 + 0$) has on the wavefront error, leading to different results with regards to the optical performance and basically simulating the effect that may be observed on real lenses in mass production where production tolerances cannot be prevented.

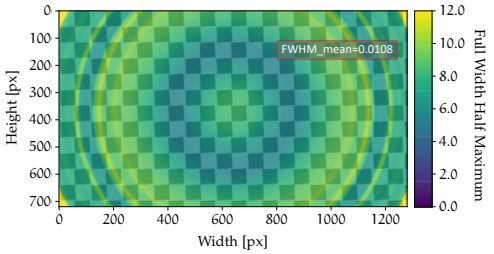
From the different parameterizations, which result in different wavefront errors, different sets of PSFs are derived via Fourier transformation. Finally, the PSFs are used to degrade the images of the datasets, which ultimately leads to seven degraded datasets for the nominal position as well as six different offsets Z_Δ between the extrema $Z_\Delta = \pm 1.25$ as shown above. To be precise, the daytime images of the BDD100K validation set are degraded for the whole range of offsets, whereas the degradation of daytime images of the BDD100K training set as well as the images of the CityScapes training and validation set are only done for the nominal positions and the extrema $Z_\Delta = \pm 1.25$. The purpose is to first show the defocus study in its entirety on small data with the BDD100K validation subset, then improve statistics in the evaluation for object detection on larger data with the BDD100K training subset but only for selected $Z_\Delta \in \{-1.25, 0, +1.25\}$, and finally extend the evaluation to instance segmentation with the CityScapes dataset for the same selected offsets.

To compare the performance of Computer Vision systems on the degraded datasets with the optical performance of the lens model and thus the spatially varying quality of images, the underlying PSFs are displayed as Full Width Half

Maximum (FWHM) maps. To this end, the directional components x and y of the two-dimensional metric are reduced to one component by taking the magnitude of both components and compute the total FWHM of a spatial position in $h \times w$ by

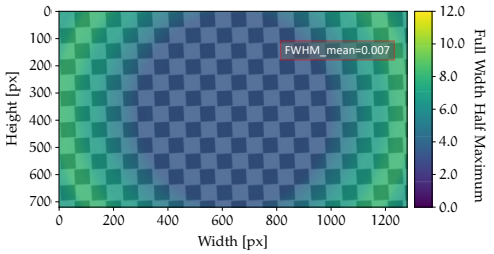
$$\mathrm{FWHM}_{total} = \sqrt{\mathrm{FWHM}_x^2 + \mathrm{FWHM}_y^2}\,.$$

Larger FWHM numbers for a spatial position in $h \times w$ indicate, simply put, that the degree of spreading in the spatially varying PSF at this point is more intense, which in turn refers to more aberration and less optical performance (see section 2.4). Figure 4.5 shows the FWHM maps from the PSFs of the optical model used to degrade images from the BDD100K dataset with parametrization $\tilde{Z}_2^0 = Z_2^0 + Z_\Delta$ for three offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ and presents in comparison one of the images in the dataset that is degraded with the respective PSF.



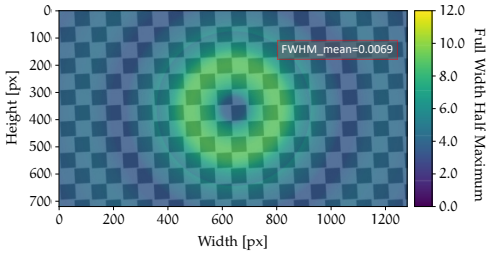
(a) FWHM map: $Z_\Delta = -1.25$

(b) Degraded image: $Z_\Delta = -1.25$

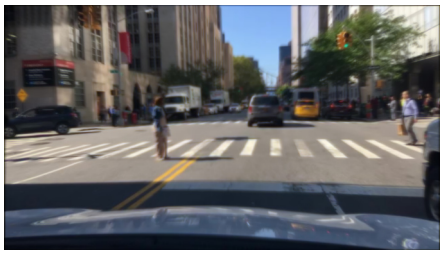


(c) FWHM map: nominal position

(d) Degraded image: nominal position



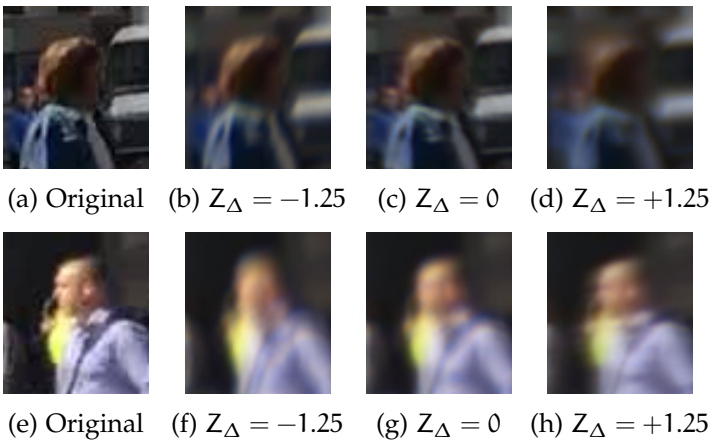
(e) FWHM map: $Z_\Delta = +1.25$



(f) Degraded image: $Z_\Delta = +1.25$

Figure 4.5: Comparison of FWHM maps and degraded images of the BDD100K dataset for defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$.

The FWHM maps in figure 4.5 are a clear indication of how the optical performance of the lens model differs spatially and where in the images the



(a) Original (b) $Z_\Delta = -1.25$ (c) $Z_\Delta = 0$ (d) $Z_\Delta = +1.25$

(e) Original (f) $Z_\Delta = -1.25$ (g) $Z_\Delta = 0$ (h) $Z_\Delta = +1.25$

Figure 4.6: Regions of interest in degraded images of the BDD100K dataset for defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ in figure 4.5 b, d, and f, respectively, in comparison with the respective region in the original image. The person in (a-d) is located at the *center* and the person in (e-h) is located at the *right side* of the images in figure 4.5.

quality is most impaired after applying the model on a given dataset. Again, larger FWHM values refer to more aberration, so warmer colors correspond to lesser optical performance. For the nominal position in figure 4.5 c (i.e. for $Z_\Delta = 0$), almost exclusively edge locations of the images are affected by the degradation. The degradation with defocus offset $Z_\Delta = +1.25$, on the other hand, affects mostly the center of the images, while the overall optical performance of the model with this parameterization is much worse compared to the nominal position, which therefore also affects edge locations. Finally, the degradation with offset $Z_\Delta = -1.25$ results in more aberration at the edges while also affecting central positions of the images.

These effects are clearly visible in the regions of interest in figure 4.6. Figure 4.6 a-d shows the person at the center and figure 4.6 f-h shows the person at the right side of the images in figure 4.5 b, d, and f, respectively. In short, the person at the center is most blurred for defocus with offset $Z_\Delta = +1.25$, whereas the quality of this region is only little affected by the model in nominal position. The person in the right corner of the image is blurred in all degraded images, but most heavily for offset $Z_\Delta = -1.25$.

In summary, the image degradation sketched in this section leads to images with spatially varying image quality. By parameterizing the optical model described in section 4.3.1 in the aforementioned way, images become, simply put, more blurred towards the edges for negative (see figure 4.5 a) and more blurred towards the center for positive (see figure 4.5 e) defocus offsets Z_Δ, while the model's nominal position basically only affects edge locations of the images. It goes without saying that increasing positive and decreasing negative offsets towards the extrema $Z_\Delta = \pm 1.25$ gradually decreases the optical performance of

the model and thus the quality of degraded images. FWHM maps are therefore only shown for $Z_\Delta \in \{-1.25, 0, +1.25\}$, from which the nominal position $Z_\Delta = 0$ represents the PSF of the optical model without manipulating Z_2^0.

Note that the image degradation of the CityScape dataset differs with respect to the image size but shows similar results in terms of the optical performance. Therefore, it is not necessary to show further FWHM maps for the underlying PSFs used to degrade images from the CityScapes dataset.

4.4 Experimental Setup

The following sections describe the experimental setup for performance evaluations of Computer Vision systems selected in section 4.2 on datasets chosen in section 4.1 in comparison with respective degraded datasets produced in section 4.3. First, in section 4.4.1, the test of object detection and instance segmentation systems on the given datasets is described. Then, in section 4.2.2, the standard performance evaluation is shown. Here, the results from the test in the first stage are used to evaluate the overall performance for different defocus conditions in comparison with the baseline performance using the Precision vs. Recall and MR vs FPPI metrics explained in section 2.3. Finally, in section 4.4.3, spatial performance evaluations with the SRI metric proposed in section 3 are described. The focus here is on statistical considerations with regards to performance evaluations in dependency of spatial positions in the input images.

4.4.1 Test on Datasets

The approach to test a Computer Vision system on a given dataset is to first convert the dataset's ground truth data into a JSON file in COCO annotation format, then run through all image names listed in the formatted JSON file and perform object detection or instance segmentation on these images while also producing an additional JSON file with results [38]. The confidence threshold of the respective system must be decreased to 0 in order not to suppress any output, so that subsequent evaluations with the Precision vs. Recall and MR vs FPPI metrics take into account the whole range of confidence thresholds (see section 2.3).

The ground truth data in COCO annotation format are listed in three different sections: "images", "categories", and "annotations". The former is a list of all image names in the dataset, which are assigned individual image IDs. For the BDD100K dataset, each image name may additionally be accompanied by labels regarding the driving scene, time of day and whether (see section 4.1). The second section, "categories", is a list of all categories labeled in the dataset. Just like the image names, the category names are assigned individual IDs, which must be done in accordance with the category IDs outputted by the Computer Vision system.

Finally, the last section, "annotations", is a list of ground truth data for individual objects in the dataset. Each individual object in the list "annotations" is assigned to an image ID representing the image name in "images" in which the object occurs, a category ID representing the category of the object in "categories", an individual annotation ID as well as bounding box coordinates and/or an instance level semantic label for object detection and instance segmentation, respectively. For the BDD100K dataset, each object may additionally be assigned a binary label for "occluded" and "truncated" in order to be able to later evaluate only specific objects with regards to their visibility.

The test on a given dataset basically runs through all image names in the ground truth data's "images" section, performs object detection and/or instance segmentation on each image, and outputs the results in a JSON file, where each output represents depending on the task either bounding box coordinates or a binary mask (or both) and comes with a score, an image ID and a category ID from which the IDs correspond to the assignments in the ground truth data.

The JSON files for ground truth data and associated object detection or instance segmentation results contain all information required to evaluate individual images and categories as shown in algorithm 2.4 and accumulate these results for further evaluations with standard metrics and the SRI.

4.4.2 Standard performance evaluation

The performance evaluation sketched here is based on the standard evaluation metrics described in section 2.3. The basic idea is as follows: First, the baseline performance on original images of a selected dataset is evaluated, next the performance evaluation is performed on the degraded datasets produced by applying the optical model with different parameterizations on these same images, and finally the performance differences between all test cases, i.e. the differences between the baseline performance and the performance on all degraded datasets, are observed. Here, the degraded datasets are differentiated by the defocus offset Z_Δ used to parameterize the underlying optical model for the image degradation.

In order to accurately monitor potential performance differences between all test cases, the concepts described in section 2.3 are used to run the evaluate for all instances as well as separately for various instance area ranges (by ignoring instances with sizes beyond the given area range). This is useful to verify, for example, if performance differences occur only for small instances. Following usual benchmarks for object detection and instance segmentation, an IoU threshold of 0.5 is used for the evaluation.

HTC with backbone ResNeXt trained on CityPersons is used for the evaluation of pedestrian detection on imaged of the BDD100K dataset. Cascade Mask R-CNN X152 trained on COCO is used for car detection and segmentation on images of the BDD100K and CityScapes dataset, respectively. For reasons mentioned

BDD100K Subset	# Fully visible pedestrians				# Cars			
	all	small	medium	large	all	small	medium	large
Train, daytime	25,450	10,524	13,098	1,828	402,222	183,163	145,243	73,816
Val, daytime	3,651	1,455	1,908	288	58,283	26,717	21,081	10,485

Table 4.3: Selection of data from the BDD100K train and val set and corresponding statistics about fully visible pedestrians and cars with respect to bounding box area ranges. Bounding box area ranges are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px. The total number of daytime images in the train and val set are 36,728 and 5,258, respectively (see table 4.1).

in section 4.2.1, only fully visible pedestrians are considered for the evaluation of HTC, while all car instances are taken into account for the evaluation of Cascade Mask R-CNN. This setup results in a solid baseline performance for both pedestrian and car detection as shown in figure 4.2 and 4.3, respectively.

First, the overall object detection performance of both systems is evaluated with the Precision vs. Recall curve on daytime images from the BDD100K validation set for the whole range of defocus offsets. Then, to improve statistics, the object detection performance is evaluated on daytime images of the BDD100K training set, but only for selected defocus offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ (i.e. extrema and nominal position). Here, the evaluation is extended to the MR vs. FPPI metric by considering the following situation: In practice, a system is deployed with a confidence threshold, whose choice in terms of the MR vs. FPPI metric is essentially a trade-off between an acceptable FP rate and a sufficiently low MR (see section 2.3.2). Simply put, the confidence threshold set for deployment refers to the choice of sensitivity for how high the score needs to be to produce a detection. It is based on the performance evaluated on certain images of datasets, but the quality of images may change with production tolerances of lenses, which is essentially what is simulated with the image degradation in section 4.3. Consequently, to evaluate the performance differences for specific choices of confidence thresholds, which, again, simulate operating points for a potential deployment, the thresholds must be based on FP rates evaluated on the original dataset. And the same confidence thresholds must be used in the evaluation on degraded images to monitor how the performance changes under effects of defocus for these particular operating points. Therefore, the choices of confidence thresholds are based on specific FPPI values from the baseline performance. For safety-critical automotive applications, the FP rate per image must be sufficiently low, so FPPI $\in \{10^{-3}, 10^{-2}, 10^{-1}\}$ are considered (cf. Pezzementi et al. [39]).

Finally, the evaluation with the standard metrics is also performed for instance

CityScapes Subset	# Pedestrians				# Cars			
	all	small	medium	large	all	small	medium	large
Train	17,395	8,980	6,770	1,640	26,180	9,228	10,226	6,718
Val	3,278	1,607	1,379	291	4,524	1,679	1,738	1,105

Table 4.4: Statistics about pedestrians and cars in the CityScapes training and validation set with respect to instance area ranges. Instance area ranges are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px. The total number of images in the train and val set are 2,975 and 500, respectively (see table 4.2).

segmentation. To this end, the CityScapes training and validation sets are used, which provide fine grained annotations for segmentation. The performance is evaluate with Cascade Mask R-CNN for the category "car". As mentioned in section 2.3, the evaluation between object detection and instance segmentation basically only differs with respect to the IoU computation. The only additional difference here is that the sizes of instances for partitioning them into different area ranges is computed on a pixel-by-pixel basis rather than by multiplying the width and height of bounding boxes.

Table 4.3 shows the statistics of instances in the BDD100K dataset that flow into the evaluation for object detection. Moreover, table 4.4 shows the instance statistics of pedestrians and cars in the CityScapes dataset. The baseline pedestrian detection performance of HTC and car detection performance of Cascade Mask R-CNN evaluated on daytime images of the BDD100K training set with the Precision vs. Recall are shown in figure 4.2 and 4.3, respectively. The instance segmentation performance of Cascade Mask R-CNN on the CityScapes training and validation set for the category "car" is shown in figure 4.4. How the overall performances with respect to the Precision vs. Recall curve and the performance for specific choices of confidence thresholds with respect to the MR vs FPPI curve change under effects of defocus is presented in section 5. The choices of confidence thresholds are in fact also important aspects for the spatial evaluation, as the SRI metric requires a threshold to suppress weak detections or segmentations. The choice of a proper confidence threshold as well as considerations for a statistically reliable SRI computation are described in section 4.4.3.

4.4.3 Spatial evaluation

In section 4.4.2, the performance evaluation of Computer Vision systems under simulated effects of defocus is based on the standard evaluation metrics. Since the effects of defocus (for these simulated defocus conditions) vary at different spatial positions in the images, the evaluation described here takes the spatial domain of

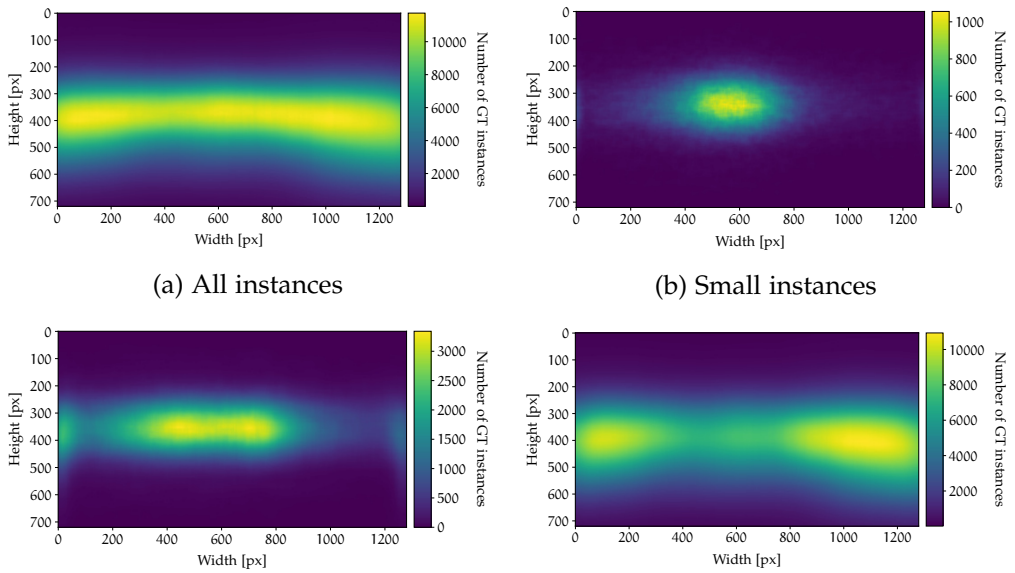
the object detection and instance segmentation performance into account. Using the SRI metric proposed in section 3, the performance is evaluated in dependency of spatial positions in the input images. Just like before in section 4.4.2, the goal is to monitor the performance differences between different test cases, which in turn are differentiated by the defocus offset Z_Δ used to parameterize the underlying optical model (see section 4.3). First, the SRI is computed for test results on original images of the datasets. Then, the SRI computation is performed for test results on degraded datasets that are produced by applying the optical model with different parameterizations in terms of Z_Δ on these same images. And finally, in order to compare the baseline performance SRI_{Base} with the performance for different defocus offsets SRI_{Z_Δ}, the spatial performance drop SRI_{Drop} is computed by

$$SRI_{Drop}(x, y) = SRI_{Base}(x, y) - SRI_{Z_\Delta}(x, y) \tag{33}$$

where x, y denote the pixels in the matrix of size $h \times w$. As shown in section 3, detections and segmentations that match a ground truth bounding box with respect to the pre-defined IoU threshold only flow into the SRI computation if they exceed a pre-defined confidence threshold. In other words, the SRI represents the performance of a Computer Vision system for a specific confidence threshold. Section 4.4.2 describes how the performance for specific operation points is evaluated with the MR vs. FPPI metric by basing the confidence threshold on FPPI values of the baseline evaluation and comparing them with performances on degraded datasets for the same thresholds. A similar approach is applied for the spatial evaluation, but instead of comparing multiple operating points as it is suggested in section 4.4.2, the evaluation with the SRI metric is performed for one specific confidence threshold, which is considered to be the threshold that best represents the overall performance of the Computer Vision system. The following paragraph describes how the threshold is chosen.

As formulated with equation 30 in section 2.3, the MR vs. FPPI curve can be summarized to a single value by averaging the MR at nine FPPI rates evenly spaced in log-space in the range 10^{-2} to 10^{0}, i.e. at FPPI= $10^{-2+\frac{x}{4}}, x \in \{0, ..., 8\}$. The result is called LAMR and gives a stable assessment of the performance analogous to the AP derived from the Precision vs. Recall curve. Since MR vs. FPPI curves are usually linear in the FPPI range from 10^{-2} to 10^{0}, the LAMR is similar to the performance at the central point between 10^{-2} to 10^{0}, i.e. at FPPI=10^{-1} [44]. Indeed, averaging the MR in the aforementioned way is a more stable assessment of the overall performance than simply picking the MR at FPPI=10^{-1}. However, since the LAMR is generally almost equal to the MR at FPPI=10^{-1}, the overall performance is best represented by the performance at the confidence threshold that corresponds to FPPI=10^{-1}.

Consequently, the choice of an appropriate confidence threshold for the SRI



(a) All instances

(b) Small instances

(c) Medium-sized instances

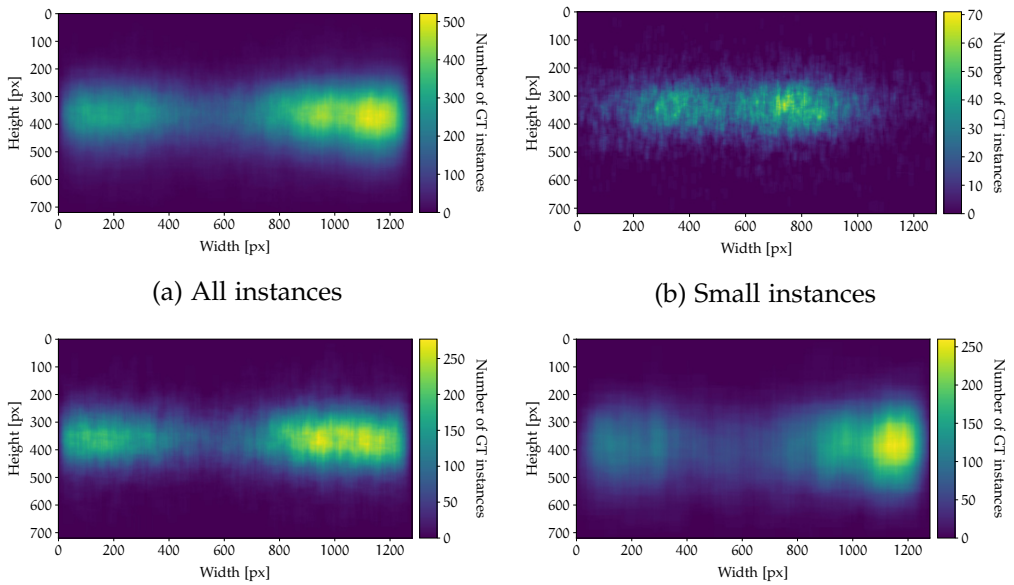
(d) Large instances

Figure 4.7: Ground truth bounding box distribution of car instances in the BDD100k training subset (images with the flag "daytime") for (a) all instances and (b-d) different bounding box area ranges. Bounding box area ranges are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

computation is based on the threshold that corresponds to FPPI=10^{-1} in the baseline evaluation. Based on the considerations in section 4.4.2, the same confidence threshold is then used to compute the SRI for the performance evaluation on degraded data rather than fixing it to FPPI=10^{-1} on that data. In other words, the chosen confidence threshold corresponds FPPI=10^{-1} in the baseline evaluation and is used to compute not only the baseline SRI but also the SRI for the degraded data. As described in section 4.4.2, this emulates a realistic scenario where a Computer Vision system is deployed with a specific choice of a confidence threshold and later subject to different optical performances of lenses, which in turn are simulated by the image degradations in section 4.3.

In summary, the confidence threshold to compute the SRI for all test cases (i.e. for the baseline performance and the performance on all degraded datasets) is based on FPPI=10^{-1} in the baseline evaluation. Besides the IoU threshold, it defines which detections are considered as TPs. Just like for the standard performance evaluation in section 4.4.2, the IoU threshold for the SRI computation is set to 0.5, which basically completes the setup for the spatial evaluation. However, there are some additional statistical aspects that need to be considered. They are discussed in the following paragraphs.

In order to be able to reliably assess the object detection or instance segmentation performance in dependency of the spatially varying optical performance,



(a) All instances

(b) Small instances

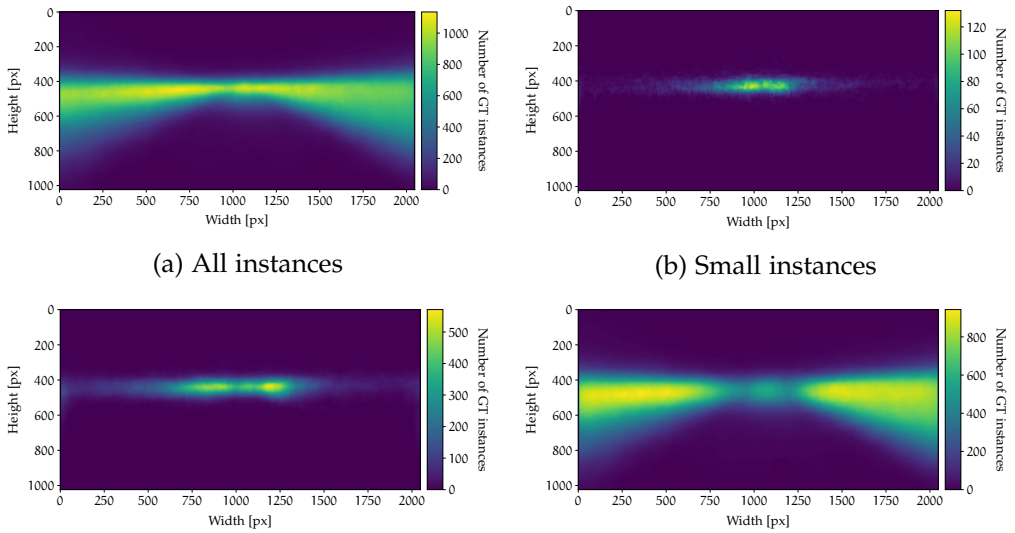
(c) Medium-sized instances

(d) Large instances

Figure 4.8: Ground truth bounding box distribution of fully visible pedestrian instances (i.e. instances without the flag "occluded" or "truncated") in the BDD100k training subset (images with the flag "daytime") for (a) all instances and (b-d) different bounding box area ranges. Bounding box area ranges are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

the number of objects must obviously be similar and sufficiently high at different locations. Furthermore, the objects' sizes must be similar at different spatial positions, because it depends largely on the size of an object how much it contributes to the SRI. This is demonstrated with the ground truth bounding box distribution of car instances in daytime images of the BDD100K training set for all, small, medium-sized and large bounding boxes in figure 4.7.

As described in section 3.1, the ground truth bounding box distribution for one category represents for each pixel x, y how often it is part of a ground truth bounding box in the given dataset. When looking at the ground truth distribution for all car instances in figure 4.7 a, the bounding boxes seem to be well distributed over the entire width of the images. Partitioning the ground truth distribution into distributions for different bounding box area ranges, however, indicates that small objects (representing distant cars) tend to be more often at the center of images (see figure 4.7 b) while larger objects (representing close cars) occur rather at the edges (see figure 4.7 d). The total number of large bounding boxes, however, is at each spatial position higher than that for small bounding boxes, which even applies to central positions where the number of large and small ground truth instance is approximately 6000 and 1000, respectively. These statistics are misleading, since according to table 4.3 there are in total 183,163



(a) All instances

(b) Small instances

(c) Medium-sized instances

(d) Large instances

Figure 4.9: Ground truth distribution of instance-level semantic labels of car instances in the CityScapes training and validation set for (a) all instances and (b-d) different instance area ranges. Instance area ranges are defined as small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

small instances but only 73,816 large instances in the BDD100K training subset. Simply put, larger bounding boxes extend over more pixels and thus contribute more to the overall ground truth distribution and subsequently to the SRI.

If, based on the considerations above, all instances were considered for the SRI computation, large instances would overshadow the results of the remaining instances even though they account for only only 73,816 of all 402,222 car instances in the subset. Therefore, the evaluation with the SRI is narrowed down to medium-sized car instances, which according to figure 4.7 c are relatively well distributed with approximately 3,000 instances at central locations and about 1,500 instances at the edge locations.

As shown in figure 4.8, the ground truth bounding box distribution for fully visible pedestrians in the dataset is similar to that for cars in the sense that small instances occur more often at the center while larger instances are rather found at the edges. The overall distribution, however, is shifted more towards the edges, indicating that pedestrians obviously rather occur on the sidewalks than on the streets. The phenomenon that larger instances overshadow the remaining ones is also represented, although not as dramatic as for cars. According to table 4.3, large instances account only for 1,828 of all 25,450 instances in the dataset, but contribute largely to the overall ground truth distribution. Medium-sized instances are again relatively well distributed with approximately 150 bounding boxes at central locations and 250 at edge locations. Consequently, the

spatial evaluation for pedestrians is, just like for cars, done by restricting the SRI computation to medium-sized instances.

For the sake of completeness, figure 4.9 shows the ground truth distribution of instance-level semantic labels of car instances in the CityScapes dataset. Just like car instances in the BDD100K dataset, small instances occur almost exclusively at the center, whereas large instances appear rather at the edges. However, in general, the cars in the CityScapes dataset are not as well distributed as cars in the BDD100K dataset. This is probably due to the large scene diversity of the BDD100K dataset, which is the reason why it is chosen in the first place. Nevertheless, the SRI computation for instance segmentation is, for the same reasons as outlined above for object detection, narrowed down to medium-sized instances.

5 Performance under effects of defocus

This section presents the results of the experiments outlined in section 4 by comparing the baseline performance of Computer Vision systems with the performance under effects of defocus. Results for object detection are presented in section 5.1 and instance segmentation results are shown in section 5.2. Moreover, in section 5.3, examples of images that show the largest change in object detection and instance segmentation performance under effects of defocus are shown.

The overall performances under effects of defocus for object detection an instance segmentation in section 5.1 and 5.2, respectively, are first visualized with standard metrics such as the Precision vs. Recall and MR vs. FPPI curves. In each plot, the baseline performance, i.e. the performance evaluated on the original data, is shown in blue, and the performances evaluated on the respective degraded datasets are assigned individual colors for different defocus offsets Z_Δ. Each plot is equipped with two legends. The legends of the Precision vs. Recall charts show the test cases (i.e. baseline, nominal, $Z_\Delta = -1.25$, etc.) and their achieved Average Precision, respectively. The items in both legends are arranged in descending order in accordance with the overall performances of the test cases, i.e. based on how well the curves approach the upper right corner of the chart or, in numeric terms, how close the Average Precision is to 1. The legends of the MR vs. FPPI curves show the test cases and confidence thresholds for specific operating points, respectively. Contrary to the Precision vs. Recall curves, lower MR vs. FPPI curves indicate better performance, so the legends are sorted in reverse order from lowest to highest overall performance.

The spatial performances under effects of defocus are displayed as heat maps. For each test case, a heat map shows the SRI difference between the baseline performance and the performance on the degraded dataset (see equation 33) with warmer colors indicating larger drop in performance. These heat maps may thus be compared with the FWHM maps presented in section 4.4.3, where warmer colors correspond to lesser optical performance of the applied lens model.

5.1 Object Detection Performance

This section presents the results for object detection. First, object detection results evaluated with standard evaluation metrics are presented in section 5.1.1. Then, results from the spatial evaluation with the SRI metric are shown in section 5.1.2.

5.1.1 Overall Performance

The results in this section are from the standard performance evaluation described in section 4.4.2. For the sake of clarity, the *evaluation setup* is summarized below:

- *IoU threshold*: 0.5
- *Test cases*: Original datasets (i.e. baseline) and datasets degraded with defocus offsets $Z_\Delta \in \{-1.25, -0.75, -0.5, 0, +0.5, +0.75, +1.25\}$
- *Data*: For the performance evaluation with all defocus offsets, daytime images of the BDD100K validation set are used. This subset accounts for $5,258$ images in the validation set. For the evaluation of the selected offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$, daytime images of the BDD100K training set are used to improve statistics. This subset accounts for $36,728$ images in the training set.
- *Detection systems*: Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) is used for car detection and HTC [1] with backbone ResNeXt is used for pedestrian detection.
- *Instances*: For pedestrian detection, only fully visible instances, i.e. pedestrians with "occluded"=False and "truncated"=False, are considered. The number of pedestrian instances in daytime images of the training and validation sets are $25,450$ and $3,651$, respectively. For car detection, all instances are considered. This makes a total of $402,222$ instances in the training subset and $58,283$ instances in the validation subset.

Figure 5.1 and 5.2 compare the baseline object detection performance with the performance under effects of defocus on the BDD100K validation set for all defocus offsets using the Precision vs. Recall metric. The former, figure 5.1, shows the car detection performance and the latter, figure 5.2, presents the pedestrian detection performance. Then, figure 5.3 and 5.4 show the baseline object detection performance in comparison with the performances for $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with the Precision vs. Recall metric for different bounding box area ranges on the BDD100K training set. Here, figure 5.3 presents the results for car detection and 5.4 shows the results for pedestrian detection. Finally, with the MR vs. FPPI metric, figure 5.5 and 5.6 compare the baseline performance for specific operating points (i.e. choices of confidence thresholds) of object detection systems with the performance at the same operating points under effects of defocus for $Z_\Delta \in \{-1.25, 0, +1.25\}$. The MR vs. FPPI curves are from the evaluation on the BDD100K training set. Figure 5.5 presents the results for car detection and figure 5.6 shows the results for pedestrian detection.

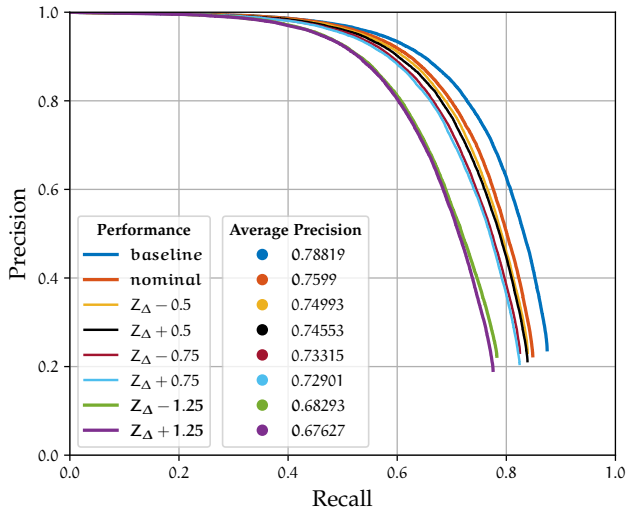


Figure 5.1: Baseline *car detection* performance in comparison with car detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, -0.75, -0.5, 0, +0.5, +0.75, +1.25\}$ evaluated with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) on daytime images of the BDD100K validation set using the Precision vs. Recall metric.

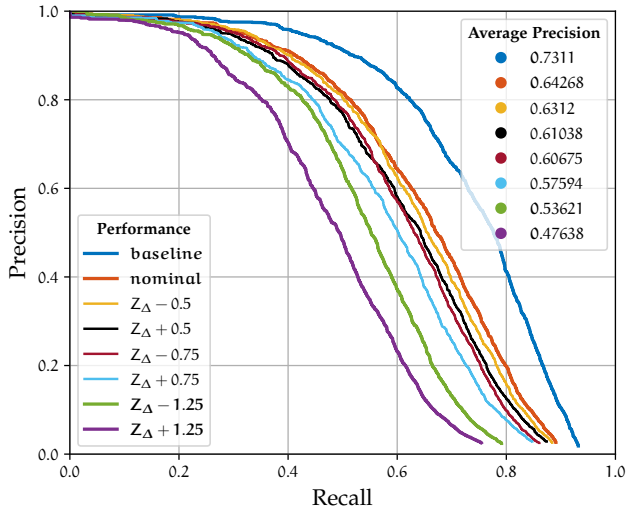
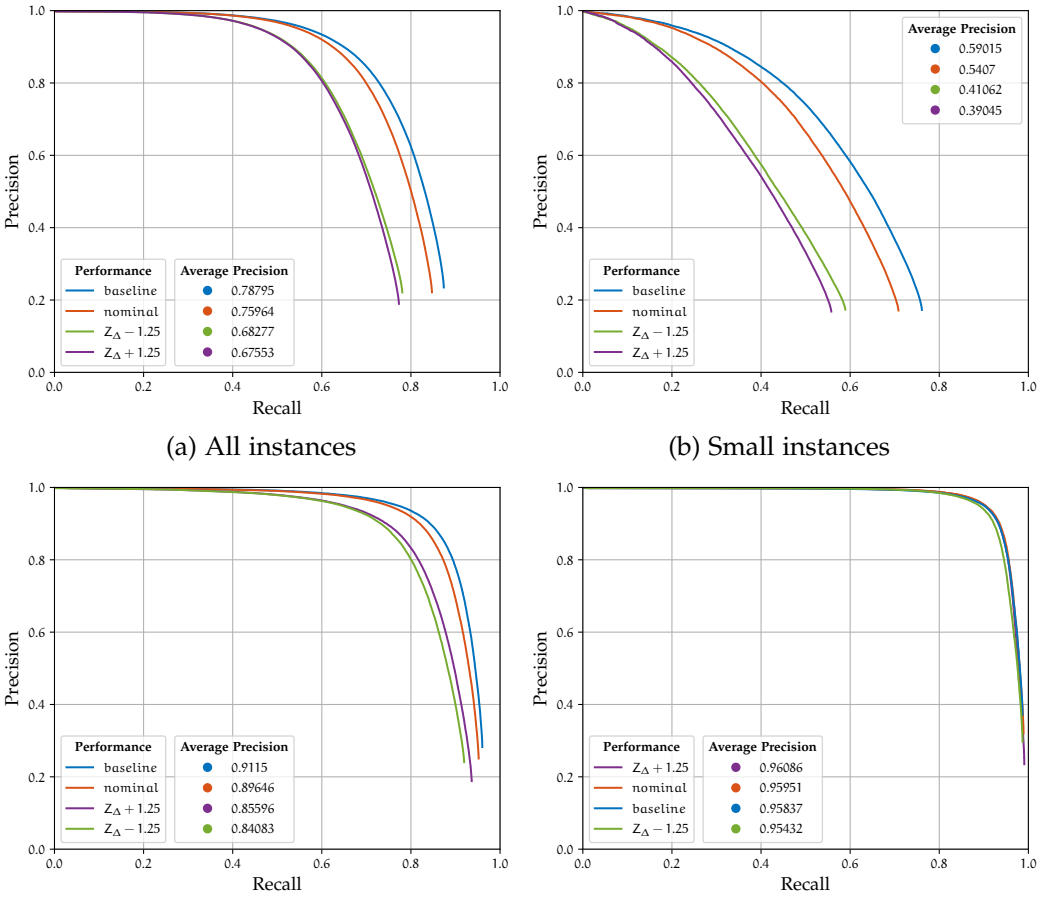


Figure 5.2: Baseline *pedestrian detection* performance in comparison with pedestrian detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, -0.75, -0.5, 0, +0.5, +0.75, +1.25\}$ evaluated with HTC [1] with backbone ResNeXt (trained on CityPersons dataset) on daytime images of the BDD100K validation set using the Precision vs. Recall metric.



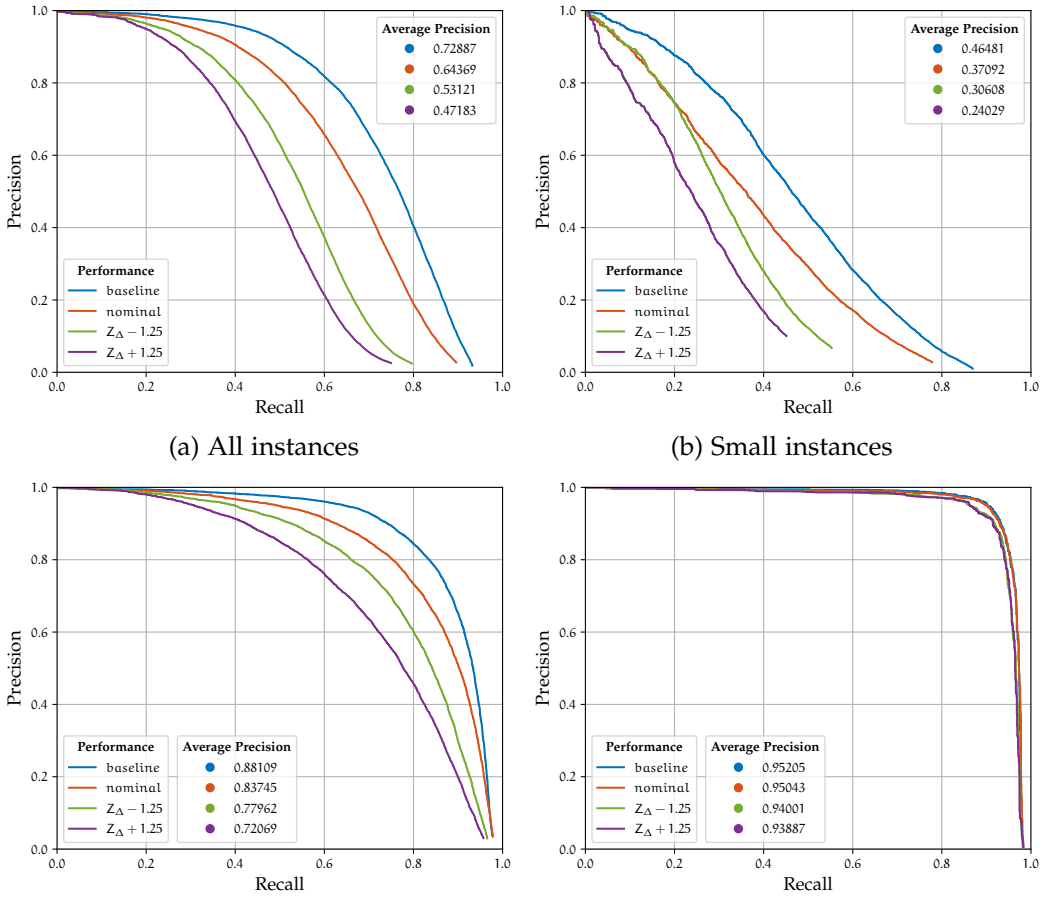
(a) All instances

(b) Small instances

(c) Medium-sized instances

(d) Large instances

Figure 5.3: Baseline *car detection* performance in comparison with car detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) on daytime images of the BDD100K training set for different bounding box area ranges using the Precision vs. Recall metric. Bounding box area ranges in are defined as (a) all $= [0, \infty]$px, (b) small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.



(a) All instances

(b) Small instances

(c) Medium-sized instances

(d) Large instances

Figure 5.4: Baseline *pedestrian detection* performance in comparison with pedestrian detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with HTC [1] with backbone ResNeXt (trained on CityPersons dataset) on daytime images of the BDD100K training set for different bounding box area ranges using the Precision vs. Recall metric. Bounding box area ranges in are defined as (a) all $= [0, \infty]$px, (b) small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

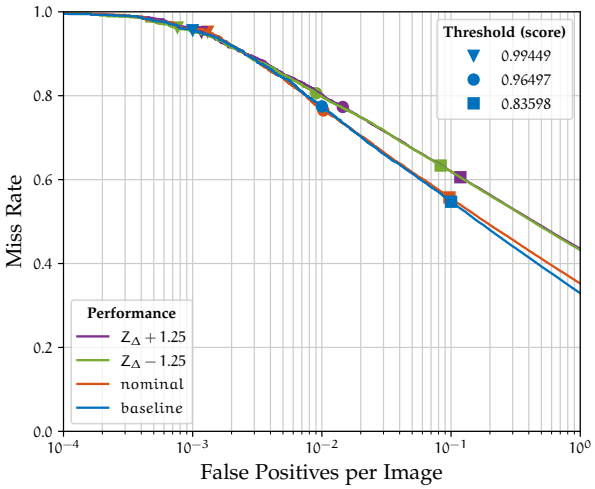


Figure 5.5: Baseline *car detection* performance in comparison with car detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) on daytime images of the BDD100K training set using the MR vs. FPPI metric. Specific operating points in terms of confidence thresholds that correspond in the baseline evaluation to FP rates of one every 10 images (rectangle), every 100 images (circle), every 1000 images (triangle) are compared.

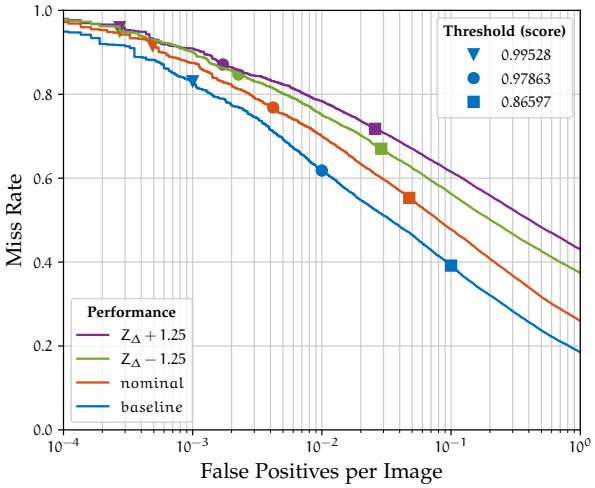


Figure 5.6: Baseline *pedestrian detection* performance in comparison with pedestrian detection performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with HTC [1] with backbone ResNeXt (trained on CityPersons dataset) on daytime images of the BDD100K training set using the MR vs. FPPI metric. Specific operating points in terms of confidence thresholds that correspond in the baseline evaluation to FP rates of one every 10 images (rectangle), every 100 images (circle), every 1000 images (triangle) are compared.

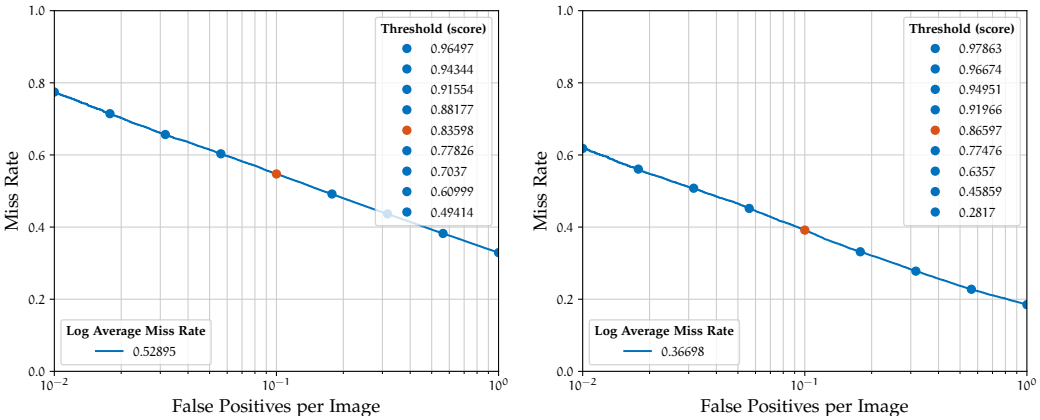
5.1.2 Spatial Performance

The results in this section are from the spatial performance evaluation described in section 4.4.3. The *evaluation setup* is summarized below:

- *IoU threshold*: 0.5
- *Confidence threshold*: The choices of confidence thresholds for HTC and Cascade Mask R-CNN are based on the MR vs. FPPI curves shown in figure 5.7. The confidence threshold for pedestrian detection is set to 0.87597 and threshold for car detection is set to 0.83598.
- *Test cases*: Original dataset (i.e. baseline) and datasets degraded with defocus offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$
- *Data*: Daytime images of the BDD100K training set. This subset accounts for 36,728 images in the training set.
- *Detection systems*: Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) is used for car detection and HTC [1] with backbone ResNeXt is used for pedestrian detection.
- *Instances*: Only medium-sized bounding boxes, i.e. bounding boxes with sizes in the range $= [32^2, 96^2]$px, are considered. Pedestrian instances are, just like in the standard evaluation, further narrowed down by considering only fully visible instances, i.e. pedestrians with "occluded"=False and "truncated"=False. In total, there are 13,098 fully visible medium-sized pedestrian instances and 145,243 medium-sized car instances in daytime images of the BDD100K training set

Figure 5.8 depicts the approach in this section. It shows exemplary the computation of the SRI for baseline car detection in comparison with the SRI for car detection under defocus with offset $Z_\Delta = -1.25$, as well as the performance drop that results from the element-wise subtraction of both SRI matrices based on equation 33. Moreover, it compares one image of the original dataset with the respective image of the degraded dataset to show on which spatial locations the image quality is most degraded.

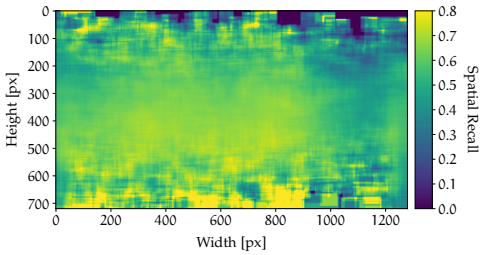
Then, figure 5.9 and 5.10 compare for each defocus offset $Z_\Delta \in \{-1.25, 0, +1.25\}$ the spatial performance drop in object detection with the respective optical performance of the parameterized lens model. More specifically, for each test case, the FWHM map presented in section 4.3.2 is compared with the corresponding heat maps of the SRI performance drop. The colors in the heat maps for the SRI performance drop and the FWHM maps are consistent in the sense that warmer colors correspond to larger drop in object detection and lesser



(a) Car detection (b) Pedestrian detection

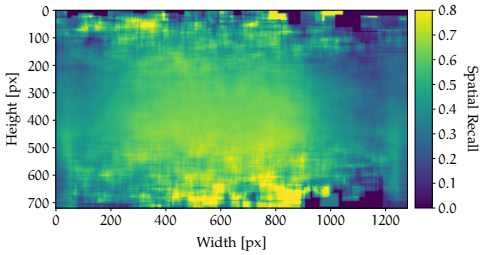
Figure 5.7: Selection of confidence thresholds with the help of the MR vs. FPPI metric evaluated with an IoU of 0.5 on daytime images of the BDD100K training set. The selected confidence thresholds for (a) *car detection* with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) and (b) *pedestrian detection* with HTC [1] with backbone ResNeXt (trained on CityPersons dataset) correspond to FPPI $= 10^{-1}$, which is the central point in the range 10^{-2} to 10^{0} where the MR vs. FPPI curves are somewhat linear. The operating point at FPPI $= 10^{-1}$ and the associated confidence threshold in the upper right legend are marked in red. The MR at this point is similar to the Log Average Miss Rate shown in the lower left legend, which, in turn, is computed by averaging the MR at all nine marked points with equation 30. According to the explanation in section 4.4.3, the confidence threshold that corresponds to FPPI $= 10^{-1}$ is considered to be most representative of the overall performance of the detection systems. These particular operating points for pedestrian detection and car detection are also marked (blue rectangle) in figure 5.5 and 5.6, respectively.

optical performance of the lens model, respectively. Figure 5.9 depicts the results for car detection and figure 5.10 shows the results for pedestrian detection.



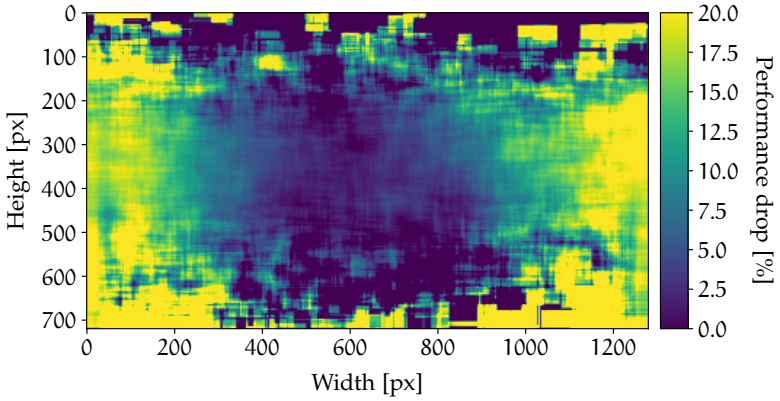
(a) Spatial Recall Index: Baseline

(b) Example image



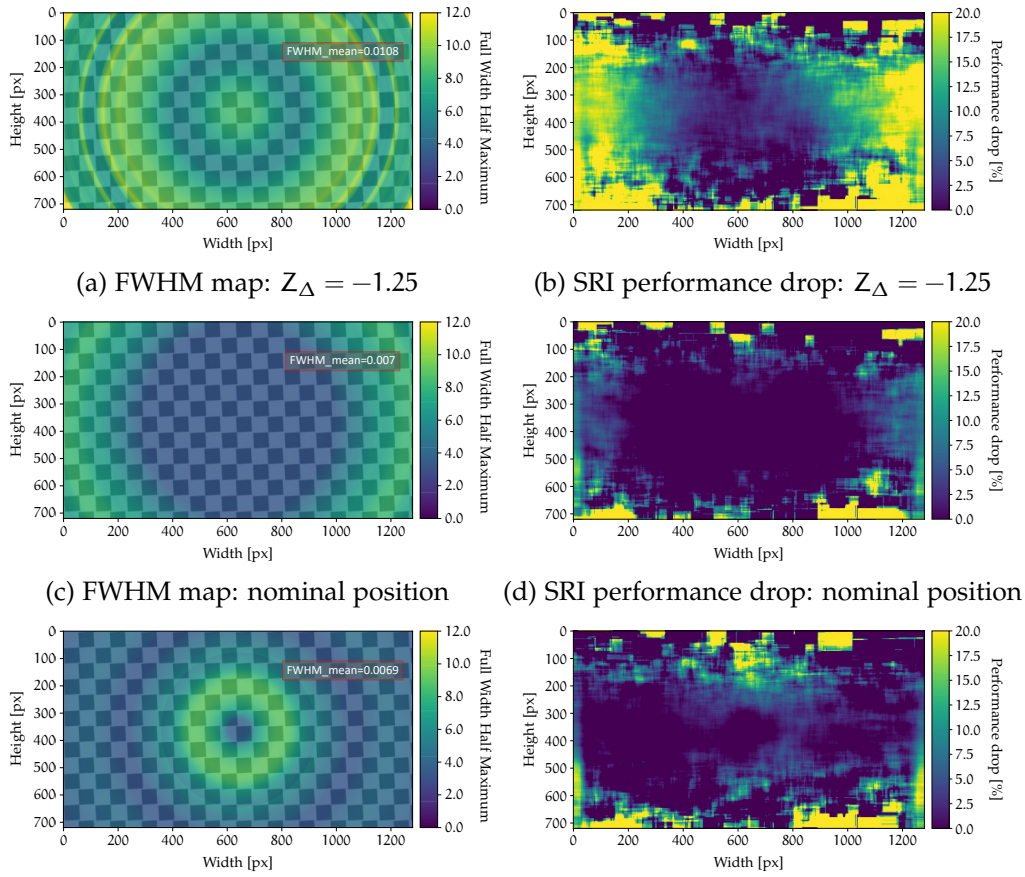
(c) Spatial Recall Index: $Z_\Delta = -1.25$

(d) Degraded image: $Z_\Delta = -1.25$



(e) Performance drop: $Z_\Delta = -1.25$

Figure 5.8: Example (a) SRI for baseline car detection with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) in comparison with the (c) SRI for car detection under defocus with offset $Z_\Delta = -1.25$, as well as the (e) performance drop that results from the element-wise subtraction of the SRI matrices based on equation 33. The spatial evaluation is carried out with an IoU threshold of 0.5 for medium-sized car instances on daytime images of the BDD100k training set. The chosen confidence threshold corresponds to FPPI $= 10^{-1}$ and is defined based on the baseline evaluation in order to simulate a realistic configuration for deployment (see selection of confidence threshold in figure 5.7 a). In order to additionally show on which spatial locations the image quality is most degraded, an (b) example image of the original dataset is compared with the respective (d) image of the degraded dataset. While the entire image is blurred, the regions at the edges are most affected by the degradation, which, in turn, is also reflected in the (c) SRI and in the (e) performance drop.



(a) FWHM map: $Z_\Delta = -1.25$

(b) SRI performance drop: $Z_\Delta = -1.25$

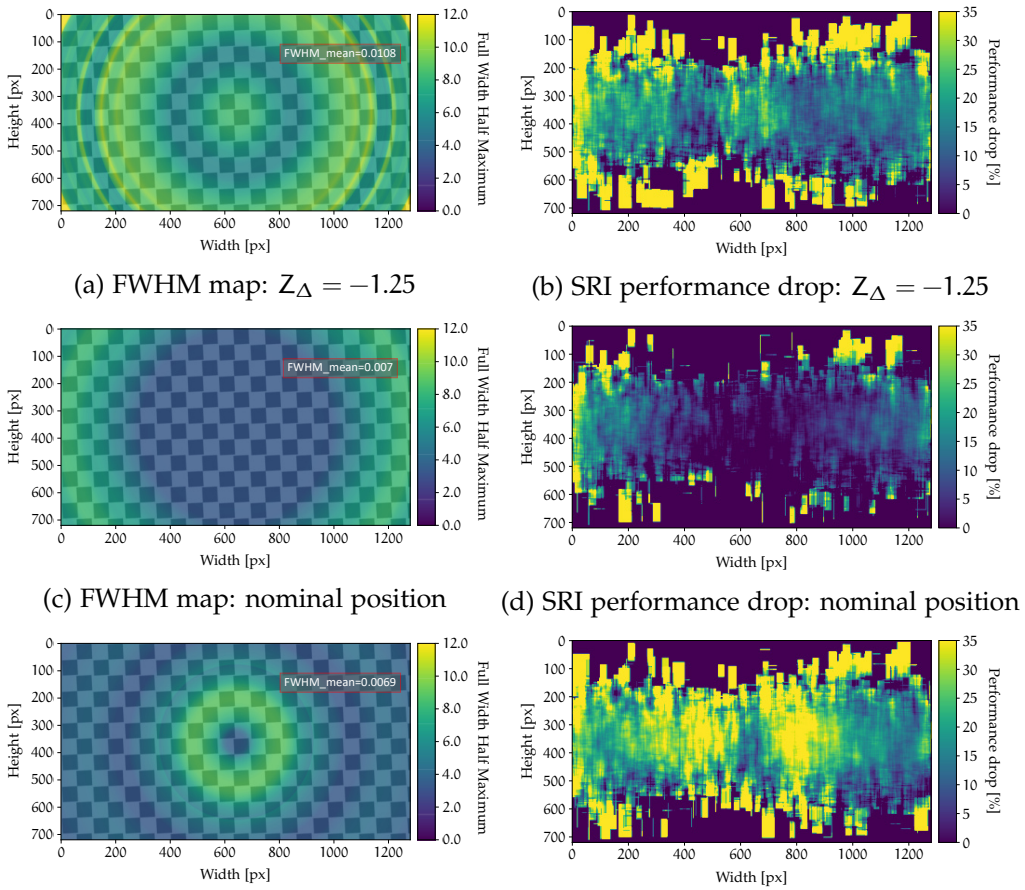
(c) FWHM map: nominal position

(d) SRI performance drop: nominal position

(e) FWHM map: $Z_\Delta = +1.25$

(f) SRI performance drop: $Z_\Delta = +1.25$

Figure 5.9: Comparison of FWHM maps and SRI performance drop for *car detection* with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$. FWHM maps on the left indicate the optical performance of the applied lens model parameterized with the respective defocus offsets Z_Δ at spatial positions of the images. The SRI performance drop on the right is calculated by subtracting the SRI computed on the dataset that is degraded by the model with respective parameterization in regards to Z_Δ from the baseline SRI computed on the original dataset (see equation 33). Warmer colors indicate lesser optical performance and larger drop in object detection performance, respectively. The spatial evaluation is carried out with an IoU threshold of 0.5 for medium-sized car instances on daytime images of the BDD100k training set. The chosen confidence threshold corresponds to FPPI $= 10^{-1}$ and is defined based on the baseline evaluation in order to simulate a realistic configuration for deployment (see selection of confidence threshold in figure 5.7 a).



(a) FWHM map: $Z_\Delta = -1.25$

(b) SRI performance drop: $Z_\Delta = -1.25$

(c) FWHM map: nominal position

(d) SRI performance drop: nominal position

(e) FWHM map: $Z_\Delta = +1.25$

(f) SRI performance drop: $Z_\Delta = +1.25$

Figure 5.10: Comparison of FWHM maps and SRI performance drop for *pedestrian detection* with HTC [1] with backbone ResNeXt (trained on CityPersons dataset) under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$. FWHM maps on the left indicate the optical performance of the applied lens model parameterized with the respective defocus offsets Z_Δ at spatial positions of the images. The SRI performance drop on the right is calculated by subtracting the SRI computed on the dataset that is degraded by the model with respective parameterization in regards to Z_Δ from the baseline SRI computed on the original dataset (see equation 33). Warmer colors indicate lesser optical performance and larger drop in object detection performance, respectively. The spatial evaluation is carried out with an IoU threshold of 0.5 for medium-sized fully visible pedestrian instances on daytime images of the BDD100k training set. The chosen confidence threshold corresponds to FPPI $= 10^{-1}$ and is defined based on the baseline evaluation in order to simulate a realistic configuration for deployment (see selection of confidence threshold in figure 5.7 b).

5.2 Instance Segmentation Performance

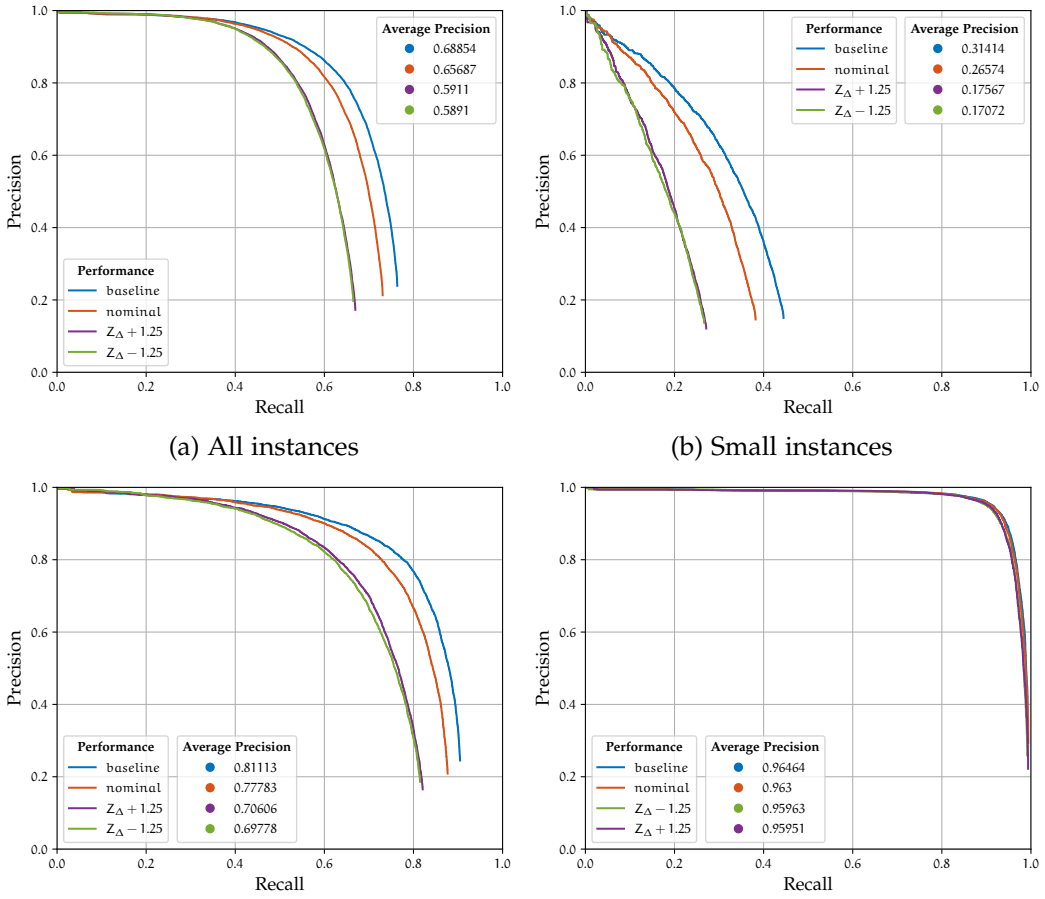
This section presents the results for instance segmentation. First, in section 5.2.1, the overall performance evaluated with the Precision vs. Recall metric is shown. Then, the spatial performance is shown in section 5.2.2.

5.2.1 Overall Performance

The evaluation setup for instance segmentation leading to the results in this section is based on the description in section 4.4.2 and can be briefly summarized as follows:

- *IoU threshold*: 0.5
- *Test cases*: Original datasets (i.e. baseline) and datasets degraded with defocus offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$
- *Data*: Combination of CityScapes training and validation set (i.e. $3,475$ images according to table 4.2)
- *Detection systems*: Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset)
- *Instances*: All car instances in the dataset (i.e. $30,704$ instances according to table 4.2)

Figure 5.11 compares the baseline instance segmentation performance in comparison with the performance under effects of defocus evaluated with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) for the category car for different instance area ranges on a combination of the CityScapes training and validation sets using the Precision vs. Recall metric.



(a) All instances

(b) Small instances

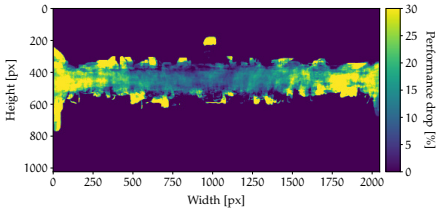
(c) Medium instances

(d) Large instances

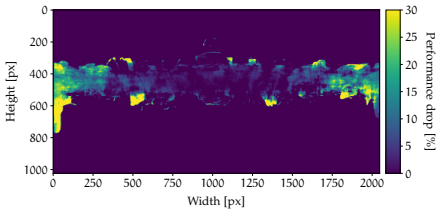
Figure 5.11: Baseline *instance segmentation* performance for the category *car* in comparison with the instance segmentation performance under effects of defocus for offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) on the CityScapes training and validation sets for different instance area ranges using the Precision vs. Recall metric. Instance area ranges in are defined as (a) all $= [0, \infty]$px, (b) small $= [0, 32^2]$px, medium $= [32^2, 96^2]$px and large $= [96^2, \infty]$px.

5.2.2 Spatial Performance

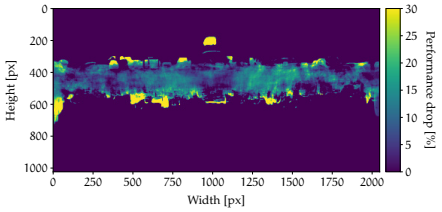
The evaluation setup for the spatial performance evaluation for instance segmentation on a combination of the CityScapes training and validation sets (i.e. 3,475 images) from the description in section 4.4.3 is as follows: The confidence threshold corresponds to FPPI $= 10^{-1}$ in the baseline evaluation, the IoU threshold is set to 0.5, test cases consist of the original dataset and degraded datasets with defocus offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$, and only medium-sized car instances are considered (i.e. 11,964 instances according to table 4.4). According to this, figure 5.12 shows the spatial performance drop for $Z_\Delta \in \{-1.25, 0, +1.25\}$.



(a) SRI performance drop: $Z_\Delta = -1.25$



(b) SRI performance drop: nominal position



(c) SRI performance drop: $Z_\Delta = +1.25$

Figure 5.12: SRI performance drop for *instance segmentation* for the category *car* with Cascade Mask R-CNN X152 [2, 6] (trained on COCO dataset) under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$. The SRI performance drop is calculated by subtracting the SRI computed on the dataset that is degraded by the model with respective parameterization in regards to Z_Δ from the baseline SRI computed on the original dataset (see equation 33). The spatial evaluation is carried out with an IoU threshold of 0.5 for medium-sized car instances on the CityScapes training and validation sets. The chosen confidence threshold corresponds to FPPI $= 10^{-1}$ and is defined based on the baseline evaluation in order to simulate a realistic configuration for deployment.

5.3 Examples with largest performance drop

After presenting the results for object detection and instance segmentation in section 5.1 and 5.2, respectively, this section shows some concrete examples for both Computer Vision tasks. The examples basically link the results from the overall evaluation with the results from the spatial evaluation by showing objects in images with the largest performance drop in the overall evaluation under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ along with its locations in the images (cf. Pezzementi et al. [39]).

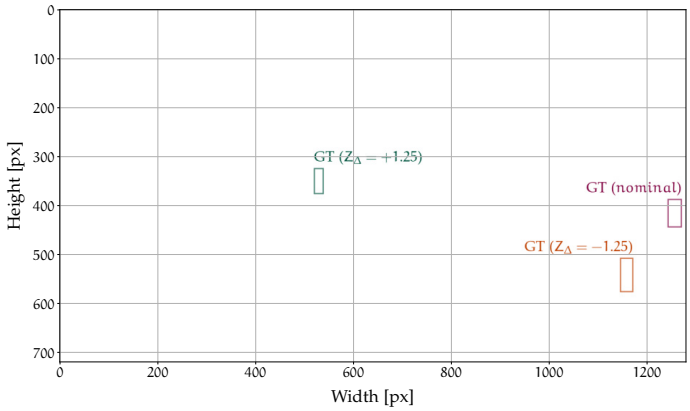
Figure 5.13 presents examples extracted from the overall evaluation with an IoU of 0.5 on fully-visible pedestrians of the BDD100K training set. In other words, the examples are extracted from the evaluation whose results are shown in figure 5.4 a and 5.6. For each test case (i.e. $Z_\Delta \in \{-1.25, 0, +1.25\}$), the example with the largest performance drop in terms of the score or FPPI under defocus along with the location in the image is shown.

Analogously, figure 5.14 presents car instances with the largest performance drop under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ from the overall evaluation for instance segmentation on the CityScape training and validation set. Consequently, the examples are extracted from the evaluation whose results are shown in figure 5.11 a. Just like before, for each test case (i.e. $Z_\Delta \in \{-1.25, 0, +1.25\}$), the example with the largest performance drop in terms of the score or FPPI is shown, while also presenting the location of the object in the image.

(a) GT instance (b) DT (baseline) (c) DT (defocus $Z_\Delta = -1.25$)

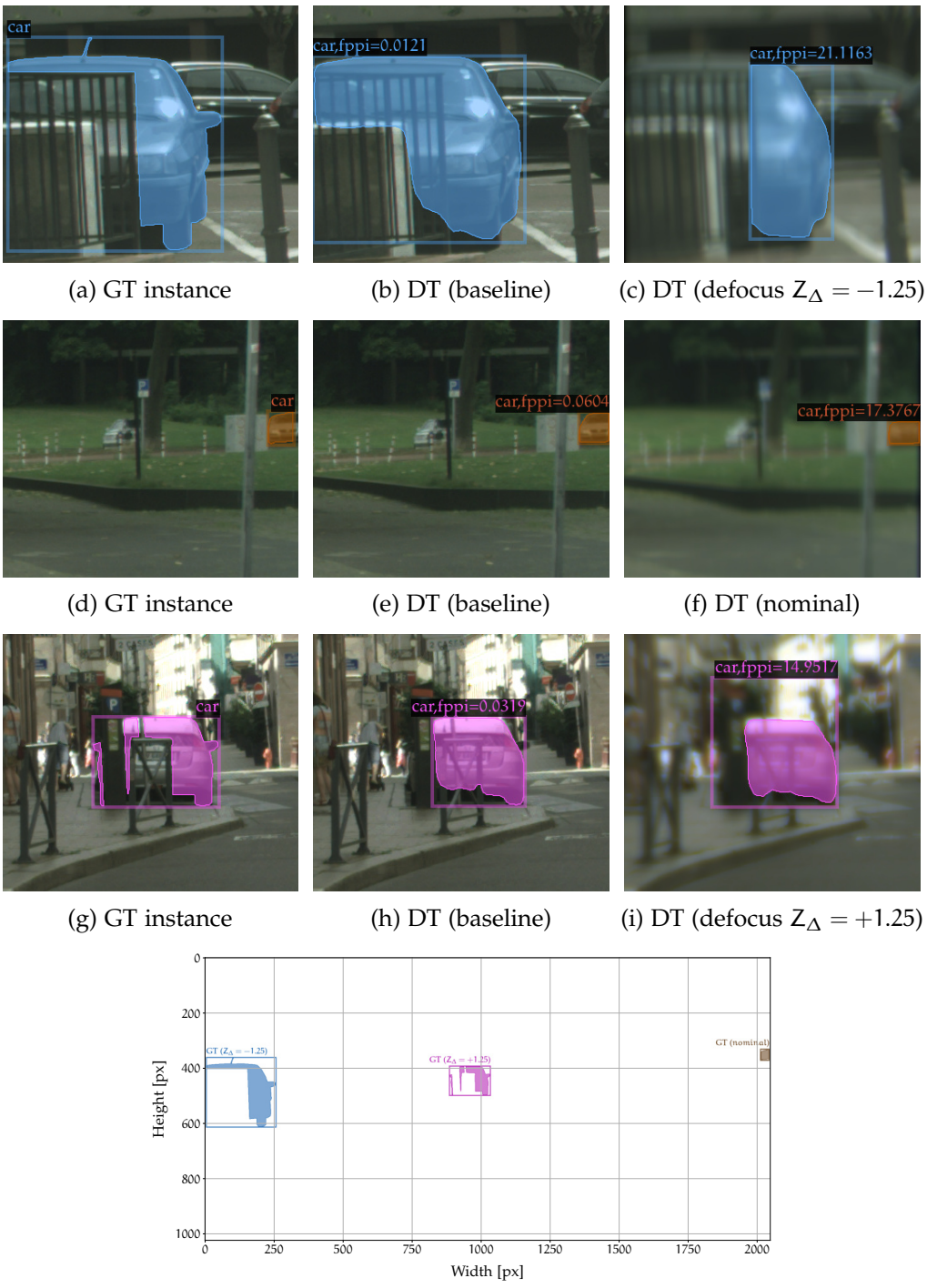
(d) GT instance (e) DT (baseline) (f) DT (nominal)

(g) GT instance (h) DT (baseline) (i) DT (defocus $Z_\Delta = +1.25$)



(j) Locations of GT instances in (a), (d), and (g)

Figure 5.13: Examples of fully visible pedestrian instances (i.e. instances without the flag "occluded" or "truncated") in the BDD100k train subset (images with the flag "daytime") that show the largest performance drop for object detection with HTC (with backbone ResNeXt) under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with an IoU threshold of 0.5. While detected very reliably with scores corresponding to FPPI rates below 0.004 in the baseline evaluation, the number of FPs per image required to still correctly detect the instances under effects of defocus ranges from approximately 3 to 14. The GT instances are located at positions where the degree of aberration is particularly extreme, i.e. for $Z_\Delta \in \{-1.25, 0\}$ at the edges and for $Z_\Delta = +1.25$ at the center of the image (see figure 5.10).



(a) GT instance (b) DT (baseline) (c) DT (defocus $Z_\Delta = -1.25$)

(d) GT instance (e) DT (baseline) (f) DT (nominal)

(g) GT instance (h) DT (baseline) (i) DT (defocus $Z_\Delta = +1.25$)

(j) Locations of GT instances in (a), (d), and (g)

Figure 5.14: Examples of car instances in the CityScapes train-val set that show the largest performance drop for instance segmentation with Cascade Mask R-CNN X152 under defocus with offsets $Z_\Delta \in \{-1.25, 0, +1.25\}$ evaluated with an IoU threshold of 0.5. While detected very reliably with scores corresponding to FPPI rates below 0.07 in the baseline evaluation, the number of FPs per image required to still correctly segment the instances under effects of defocus ranges from approximately 15 to 21. The GT instances are located at positions where the degree of aberration is particularly extreme, i.e. for $Z_\Delta \in \{-1.25, 0\}$ at the edges and for $Z_\Delta = +1.25$ at the center of the image.

6 Discussion and conclusion

After an introduction into the fields of Computer Vision and Optics, this work has linked both research areas by evaluating the spatial dependency of the performances of object detection and instance segmentation systems on the spatially varying performance of optical systems. To this end, a newly proposed evaluation metric called Spatial Recall Index was presented, which assesses the performance of these Computer Vision systems in dependency on spatial positions in input images. Briefly summarized, the link between Computer Vision and Optics was provided by first simulating a real lens that shows spatially varying optical performance, then applying the underlying optical model on large-scale Autonomous Driving datasets to simulate real-world driving scenes under naturally occurring effects of defocus, and finally comparing the overall and spatial performance drop of Computer Vision systems under these effects with the spatially varying optical performance of the simulated lens. Given the results presented in section 5, section 6.1 discusses them in detail. Finally, based on the experiments and results, the conclusion is given in section 6.2.

6.1 Discussion

First, on a small subset of the BDD100K validation set consisting of $5,258$ images with driving scenes at daytime, the overall pedestrian detection and car detection performance are contrasted under effects of defocus for small changes in the parametrization of the applied optical model and thus small changes in the quality of the degraded images towards the extrema. The changes in optical performance are represented by different defocus parameters of the optical model with $Z_\Delta = 0$ denoting the unparameterized model (i.e. nominal position) and $Z_\Delta = \pm 1.25$ indicating the extrema. While car detection with Cascade Mask R-CNN happens to be more robust to the defocus conditions than pedestrian detection with HTC, it can be seen that even moderate image mutations result in a drop in performance for both detection tasks (cf. figure 5.1 and 5.2). The AP for car and pedestrian detection drops by 3.59% and 12.09%, respectively, when applying the unparameterized optical model and the performance drop concerning the AP goes up to 14.2% and 13.35% for car detection and 34.8% and 26.66% for pedestrian detection when changing the defocus parameter to $Z_\Delta = \pm 1.25$.

Then, the pedestrian and car detection performance under effects of defocus are assessed more detailed on larger data using a subset of the BDD100K training set with $36,728$ images at daytime, while limiting the evaluation to selected test

cases consisting of the nominal position and the extrema $Z_\Delta = \pm 1.25$. Statistically, this subset contains enough labeled objects to partition the evaluation into different instance area ranges, which shows that the largest performance drop in both tasks occur for small and thus more distant instances (cf. figure 5.3 and 5.4). Applying the unparameterized optical model on the given training subset and increasing the offset to $Z_\Delta = +1.25$ reduces the AP for the detection of small car and pedestrian instances by $8,38\% - 33,84\%$ and $20,2\% - 48,3\%$, respectively, whereas the detection of large objects is not affected by the image degradation. The performance drop for medium-sized objects lies somewhere in between with a decrease in the AP of $1.65\% - 7.75\%$ for car detection and $4.95\% - 18.2\%$ for pedestrian detection.

With the MR vs. FPPI metric, the object detection performance of specific operating points are evaluated by standardizing the confidence threshold on expected FP rates per image (cf. figure 5.5 and 5.6). The baseline MR for pedestrian detection is at each operating point smaller than for car detection, which is probably because the number of cars in the dataset is more than an order of magnitude higher than the number of pedestrians (see table 4.3). The tendency that car detection is more robust to image perturbations, however, is also found in this metric, especially for a confidence threshold that corresponds to FPPI $= 10^{-1}$ in the baseline evaluation (cf. blue rectangle in figure 5.5 and 5.6).

This particular confidence threshold is used for the spatial evaluation of the car and pedestrian detection performance, where the computations are limited to medium-sized instances to remove the size-dependency of objects on the SRI. With enough well distributed objects, the resulting spatial performance drop in object detection shows a strong correlation with the optical performance of the underlying optical model and thus the spatially varying image quality, which is especially visible for car detection in figure 5.9 where $145,243$ instances flow into the evaluation. When comparing the FWHM maps with the SRI performance drop, it can be seen that for each test case, the axially symmetric behavior of the simulated lens with the respective defocus offsets is reflected on the spatial performance drop of the car detection. The nominal position of the optical model affects the detection performance virtually only at the edges of images at regions where the lens shows lesser optical performance (cf. figure 5.9 c and d). Similarly, applying the optical model with a defocus offset $Z_\Delta = -1.25$ mostly impairs the object detection performance at the edges with a gradually decreasing intensity towards the center, which is similarly reflected in the optical performance of the lens represented by the respective FWHM map (cf. figure 5.9 a and b). Finally, with a defocus offset $Z_\Delta = +1.25$, the object detection performance drops symmetrically around center of the images, which is comparable with the pattern in the respective FWHM map representing the optical performance of the simulated lens with the respective parametrization (cf. figure 5.9 e and f).

Analogously, the pedestrian detection performance is compared with the optical performance of the applied lens model in figure 5.10. Just like the standard metrics before, the spatial evaluation shows that pedestrian detection with HTC is less robust to the simulated defocus conditions than car detection with Cascade Mask R-CNN. Since the number of considered pedestrians instances in the dataset is with $13,098$ much lower than the number of car instances, the SRI and thus the spatial performance drop can not be assessed as accurately as that of car detection. However, a similar effect of the defocus conditions on the object detection performance can be observed for the nominal position, where the performance drop occurs mostly at the edges (cf. figure 5.10 c and d), and for $Z_\Delta = +1.25$, where the performance drop occurs mostly around the center (cf. 5.10 e and f). Note that, in general, most of the medium-sized car and pedestrian instances occur between the height 200px and 500px of images, so the results in this area are statistically most reliable (cf. figure 4.7 and 4.8).

Finally, with the CityScapes dataset, the evaluation is extended to instance segmentation for the category car. The overall performances evaluated with the Precision vs. Recall curves are in virtually all test cases worse than the associated car detection performance (cf. figure 5.3 and 5.11). A reason for this could be that instance segmentation is a more challenging task than object detection (see figure 4.4). For the spatial evaluation, the number of medium-sized car instances is with $10,226$ not sufficient for a reliable SRI computation, especially considering their spatial distribution in the dataset (see figure 4.9). However, the tendency that the optical model in nominal position and with defocus offset $Z_\Delta = -1.25$ affects the system's performance rather at the edges of images and applying the optical model with defocus offset $Z_\Delta = +1.25$ impairs the performance more around the center can also be observed for instance segmentation (see figure 5.12).

The same observation can be made in figure 5.13 and 5.14, where examples with the largest drop in performance are shown for object detection and instance segmentation, respectively. The examples represent TP instances with largest score differences between baseline and evaluation under defocus. All of these instances occur at spatial positions of images where the optical performance of the applied lens model with the respective parameterizations are especially poor. For most of these instances, the number of FPs per image required to still detect (or segment) them under effects of defocus increases by approximately three orders of magnitude.

6.2 Conclusion

In this work, a thorough evaluation of the dependency of Computer Vision systems on the optical performance of optical systems was provided. To this end, a newly proposed evaluation metric called Spatial Recall Index was presented,

which evaluates the object detection and instance segmentation performance in dependency on spatial locations in input images. By simulating spatially varying effects of defocus with a lens model and evaluating Computer Vision systems under these effects with both the newly proposed metric and standard evaluation metrics, a strong correlation between the performance of Computer Vision systems and the optical performance of the applied lens model and thus the optical quality of input images could be observed. More specifically, a parameterized optical model based on Zernike Polynomials was applied on the BDD100K and CityScapes datasets to simulate different effects of defocus and two Computer Vision systems were used to evaluate the performance in object detection and instance segmentation under these simulated effects. While the overall performance evaluated with standard evaluation metrics gradually decreases with small degradations in the optical quality of images, the results from the spatial evaluation show that these performance drops are reflected quite precisely by the spatially varying optical performance of the underlying optical model.

This applies to all considered Computer Vision tasks consisting of car and pedestrian detection as well as for instance segmentation. However, the clearest indication of a correlation between the spatially varying optical performance of the simulated lens with different defocus parameterizations and the spatial performance drop of Computer Vision systems was found by evaluating the car detection performance on driving scenes of the BDD100K dataset with 145,243 spatially relatively well distributed and almost equally sized car instances. With this evaluation setup, the size dependency of the instances on the SRI computation was almost completely removed while also providing enough data for a statistically reliable evaluation under different defocus conditions. The axially symmetric optical performances of the underlying, physically realistic, lens model with different defocus parameterizations (shown with FWHM maps) are for all test cases comparable with the spatial performance drops evaluated with the object detection system after applying the model on the given dataset (shown with heat maps of the SRI performance drop).

For different simulated defocus conditions, cases could be shown where the object detection and instance segmentation performance under effects of defocus drops dramatically, even under moderate image perturbations. This underlines the importance of taking into account the spatial domain when developing and evaluating Computer Vision systems, especially for safety-critical applications such as self-driving cars. To this end, section 7 describes possible future work in linking the image quality to the performance of Computer Vision algorithms.

7 Future work

This work has shown that there is a clear correlation between the performance of object detection and instance segmentation systems and the quality of images. The performance of two pre-trained systems, HTC and Cascade Mask R-CNN, dropped under simulated effects of defocus, and the performance drop correlates spatially with the spatially varying optical performance of the applied lens model. This highlights the need for further research in linking the performance of Computer Vision systems and the image quality of their inputs by taking into account the spatial domain. Moreover, it underlines the significance to evaluate not only the performance of Computer Vision systems but also their robustness to naturally occurring phenomena such as effects of defocus, especially for safety-critical applications such as self-driving cars.

The performance under effects of defocus was evaluated for two systems that are trained on two different datasets for different tasks. However, it would also be desirable to include more state-of-the-art Computer Vision systems, train them on the same datasets, and evaluate their performance in the same task under effects of defocus, as this would allow a statistically reliable comparison of their robustness to naturally occurring effects of defocus.

More importantly, further research should focus on improving the robustness of Computer Vision systems to defocus conditions. One approach may be to include degraded images into the training process and evaluate the change in robustness that the modified training yields.

Finally, the evaluation of Computer Vision systems in dependency of optical parameters should be extended to more tasks. This work focussed on object detection and instance segmentation, which involve an assignment of class-specific bounding boxes and instance-level semantic labels to objects in images. The newly proposed SRI metric may be extended to panoptic segmentation, which involves not only the assignment of instance-level semantic labels to all foreground objects but also the classification of background, i.e. *stuff*, on a per-pixel level [3].

A Appendix

Forward pass in matrix form

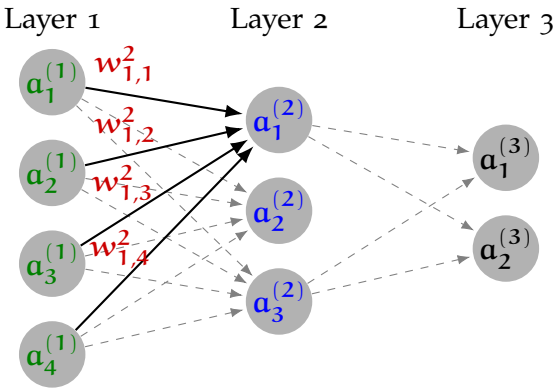


Figure A.1: Fully-connected Neural Network.

Computing the weighted sums of neurons in the first hidden layer:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{34}$$

$$z^{(2)} = W^{(2)} a^{(1)} + b^{(2)} \tag{35}$$

$$\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} & w_{1,4}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} & w_{2,3}^{(2)} & w_{2,4}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} & w_{3,3}^{(2)} & w_{3,4}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix} \tag{36}$$

Applying the non-linear activation function to compute the activations in the first hidden layer:

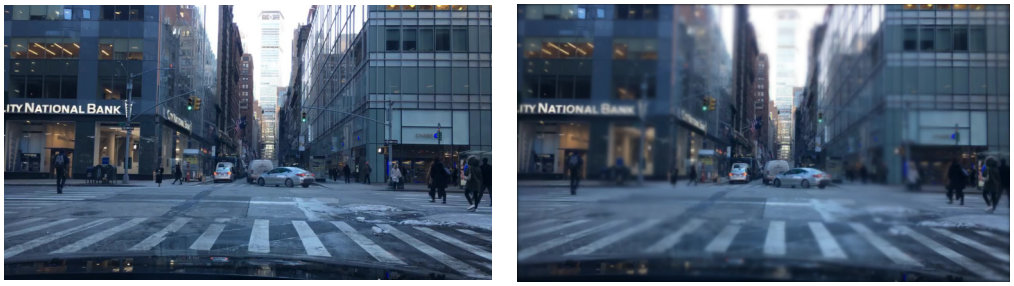
$$a^{(l)} = h\left(z^{(l)}\right) \tag{37}$$

$$a^{(2)} = h\left(z^{(2)}\right) \tag{38}$$

$$\begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} = h\left(\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}\right) \tag{39}$$

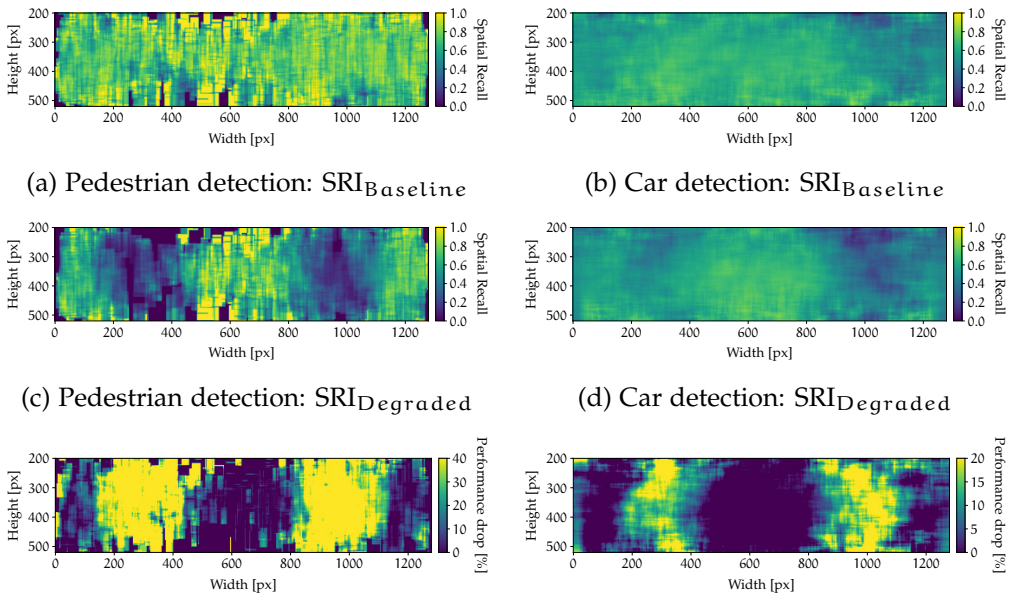
Demonstration of the SRI metric

This section demonstrates the SRI metric on extremely degraded images. Figure A.2 compares one original image with the degraded image and figure A.3 presents the SRI of Cascade Mask R-CNN for car detection and HTC for pedestrian detection. Evaluation setup: Daytime images of the BDD100K validation set (5,258 images), IoU threshold = 0.5, confidence threshold standardized to FPPI = 10^{-1}, medium-sized car instances, medium-sized fully visible pedestrian instances.



(a) Original (b) Degraded

Figure A.2: Comparison of an original image and an extremely degraded image of the BDD100K validation set for test purposes.



(a) Pedestrian detection: $\mathrm{SRI}_{\mathrm{Baseline}}$ (b) Car detection: $\mathrm{SRI}_{\mathrm{Baseline}}$

(c) Pedestrian detection: $\mathrm{SRI}_{\mathrm{Degraded}}$ (d) Car detection: $\mathrm{SRI}_{\mathrm{Degraded}}$

(e) Pedestrian detection: $\mathrm{SRI}_{\mathrm{Drop}}$ (f) Car detection: $\mathrm{SRI}_{\mathrm{Drop}}$

Figure A.3: Test of the SRI metric with extremely degraded images of the BDD100K validation set.

References

[1] Kai Chen et al. "Hybrid Task Cascade for Instance Segmentation." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 2575-7075. June 2019, pp. 4969–4978. DOI: 10.1109/CVPR.2019.00511.

[2] Zhaowei Cai and Nuno Vasconcelos. "Cascade R-CNN: High Quality Object Detection and Instance Segmentation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.5 (Nov. 2019). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1483–1498. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2019.2956516.

[3] Richard Szeliski. *Computer Vision*. Texts in Computer Science. London: Springer London, 2021. ISBN: 978-1-84882-934-3 978-1-84882-935-0. DOI: 10.1007/978-1-84882-935-0. URL: http://link.springer.com/10.1007/978-1-84882-935-0 (visited on 06/06/2021).

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org/ (visited on 06/06/2021).

[5] Marius Cordts. "Understanding Cityscapes: Efficient Urban Semantic Scene Understanding." en. PhD thesis. Darmstadt: Technische Universität Darmstadt, 2017.

[6] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. en. Publisher: Determination Press. 2015. URL: http://neuralnetworksanddeeplearning.com (visited on 06/06/2021).

[8] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks." In: *ArXiv e-prints* (Nov. 2015).

[9] Vinod Nair and Geoffrey E Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." en. In: *ICML* (2010), p. 8.

[10] Andrej Karpathy. "Connecting Images and Natual Language." en. PhD thesis. Stanford University, 2016.

[11] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 06/23/2021).

[12] Fei-Fei Li and Kevin Zakka. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/convolutional-networks/ (visited on 06/06/2021).

[13] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: *arXiv:1603.07285 [cs, stat]* (Jan. 2018). arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285 (visited on 06/25/2021).

[14] Zhong-Qiu Zhao et al. "Object Detection With Deep Learning: A Review." In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (Nov. 2019). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 3212–3232. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2018.2876865.

[15] Afzal Godil et al. "Performance Metrics for Evaluating Object and Human Detection and Tracking Systems." en. In: *NIST* (), p. 16.

[16] Licheng Jiao et al. "A Survey of Deep Learning-Based Object Detection." In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 128837–128868. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2939201.

[17] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2016.2577031. URL: http://ieeexplore.ieee.org/document/7485869/ (visited on 06/06/2021).

[18] Kaiming He et al. "Mask R-CNN." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[19] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[20] Jasper Uijlings et al. "Selective Search for Object Recognition." In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[21] Ross Girshick. "Fast R-CNN." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Dec. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[22] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 346–361. ISBN: 978-3-319-10578-9. DOI: 10.1007/978-3-319-10578-9_23.

[23] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." en. In: *ICLR* (Apr. 2015). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 06/30/2021).

[24] Kaiming He et al. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[25] Ross Girshick. *rbgirshick/py-faster-rcnn*. original-date: 2015-09-25T21:04:08Z. July 2021. URL: https://github.com/rbgirshick/py-faster-rcnn (visited on 07/07/2021).

[26] Li Fei-Fei and Justin Johnson. *CS231n Lecture 8 - Localization and Detection*. 2016. URL: https://www.youtube.com/watch?v=_GfPYLNQank&t=3126s (visited on 07/07/2021).

[27] Zhaowei Cai and Nuno Vasconcelos. "Cascade R-CNN: Delving Into High Quality Object Detection." en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 6154–6162. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00644. URL: https://ieeexplore.ieee.org/document/8578742/ (visited on 06/08/2021).

[28] Zachary Pezzementi et al. "Comparing Apples and Oranges: Off-Road Pedestrian Detection on the NREC Agricultural Person-Detection Dataset." In: (July 2017).

[29] Li Fei-Fei, Justin Johnson, and Serena Yeung. *Lecture 2 | Image Classification*. 2017. URL: https://www.youtube.com/watch?v=0oUX-nOEjG0 (visited on 07/13/2021).

[30] P. Dollar et al. "Pedestrian Detection: An Evaluation of the State of the Art." en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.4 (Apr. 2012), pp. 743–761. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2011.155. URL: http://ieeexplore.ieee.org/document/5975165/ (visited on 06/06/2021).

[31] A Geiger et al. "Vision meets robotics: The KITTI dataset." en. In: *The International Journal of Robotics Research* 32.11 (Sept. 2013), pp. 1231–1237. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913491297. URL: http://journals.sagepub.com/doi/10.1177/0278364913491297 (visited on 07/10/2021).

[32] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding." en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 3213–3223. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.350. URL: http://ieeexplore.ieee.org/document/7780719/ (visited on 06/01/2021).

[33] Shanshan Zhang, Rodrigo Benenson, and Bernt Schiele. "CityPersons: A Diverse Dataset for Pedestrian Detection." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. July 2017, pp. 4457–4465. DOI: 10.1109/CVPR.2017.474.

[34] Markus Braun et al. "EuroCity Persons: A Novel Benchmark for Person Detection in Traffic Scenes." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8 (Aug. 2019). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1844–1861. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2019.2897684.

[35] Pei Sun et al. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset." In: *arXiv:1912.04838 [cs, stat]* (May 2020). arXiv: 1912.04838. URL: http://arxiv.org/abs/1912.04838 (visited on 07/13/2021).

[36] Fisher Yu et al. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 2575-7075. June 2020, pp. 2633–2642. DOI: 10.1109/CVPR42600.2020.00271.

[37] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge." en. In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4 (visited on 06/27/2021).

[38] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context." In: *arXiv:1405.0312 [cs]* (Feb. 2015). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (visited on 06/01/2021).

[39] Zachary Pezzementi et al. "Putting Image Manipulations in Context: Robustness Testing for Safe Perception." en. In: *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. Philadelphia, PA: IEEE, Aug. 2018, pp. 1–8. ISBN: 978-1-5386-5572-6. DOI: 10.1109/SSRR.2018.8468619. URL: https://ieeexplore.ieee.org/document/8468619/ (visited on 06/06/2021).

[40] Eugene Hecht. *Optik*. de. Publication Title: Optik. De Gruyter, Mar. 2018. ISBN: 978-3-11-052665-3. URL: https://www.degruyter.com/document/doi/10.1515/9783110526653/html (visited on 07/14/2021).

[41] José Sasián. "Control of Linear Astigmatism Aberration in a Perturbed Axially Symmetric Optical System and Tolerancing." en. In: *Applied Sciences* 11.9 (Jan. 2021). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, p. 3928. DOI: 10.3390/app11093928. URL: https://www.mdpi.com/2076-3417/11/9/3928 (visited on 07/15/2021).

[42] Patrick Müller, Matthias Lehmann, and Alexander Braun. "Optical quality metrics for image restoration." en. In: *SPIE Digital Optical Technologies* (2019).

[43] Olga Kalinkina, Tatyana Ivanova, and Julia Kushtyseva. "Wavefront Parameters Recovering by Using Point Spread Function." en. In: *Proceedings of the 30th International Conference on Computer Graphics and Machine Vision (GraphiCon 2020). Part 2* (Dec. 2020), short47–1–short47–7. DOI: 10.51130/graphicon-2020-2-4-47. URL: http://ceur-ws.org/Vol-2744/short47.pdf (visited on 07/15/2021).

[44] Irtiza Hasan et al. "Generalizable Pedestrian Detection: The Elephant In The Room." In: *arXiv:2003.08799 [cs]* (Dec. 2020). arXiv: 2003.08799. URL: http://arxiv.org/abs/2003.08799 (visited on 06/05/2021).

[45] Robert Stojnic et al. *Papers with Code - Pedestrian Detection*. en. URL: https://paperswithcode.com/task/pedestrian-detection (visited on 08/09/2021).