Peter LORENZ, BSc.

A Deep-Learning Approach for Occlusion Detection

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor
Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock
Institute for Computer Graphics and Vision

Advisor
Dipl.-Ing. Patrick Knöbelreiter
Institute for Computer Graphics and Vision

Graz, Austria, March 2019

Dedicated to my Father.

All models are wrong, but some are useful.

George E. P. Box, Statistician

Abstract

We investigate the problem of occlusion detection in stereo images by evaluating three different supervised and end-to-end Convolutional Neural Network (CNN) architectures. Occlusions arise if a scene is capture by two cameras from different positions. Occluded pixels do not have any corresponding pixel in the other image and are therefore erroneous pixels. The detection of occlusions would improve other approaches relying on the corresponding problem, such as computing disparity maps or excluding occluded pixels in the loss of a neural network.

Each architecture takes rectified stereo image pairs as input. Unary CNNs for the left and right input images compute the pixel-wise similarity of the image pairs. The three models differ in how they combine the output of the unary CNNs to detect occlusions. One model learns occlusions with dilated convolutional unary CNNs. The other three learn occlusions with the stereo correlation, where the stereo correlation is refined with either 2D dilation layers or 3D convolutional layer.

We have made experiments with pre-training the models with the synthetic Sceneflow (Monkaa) dataset. We fine-tune our methods to the smaller Middlebury dataset and uses the Adam optimizer to train it. Our proposed models achieve an accuracy score between 77.98% and 89.02% on the Middlebury validation set.

Keywords: Occlusion Detection, Binary Classification, Optimization, Stereo Correlation.

Kurzfassung

Wir untersuchen das Problem der Verdeckungen in Stereobildpaaren indem wir drei verschiedene faltende neuronale Netzwerke (CNN) Architekturen auswerten. Verdeckte Pixel haben keinen korrespondierende Pixel im anderen Bild und sind somit ein fehlerhafte Pixel. Das Wissen über Verdeckungen ist hilfreich um andere Ansätze zu verbessern, welche die Berechnung der Punktkorrepondenzen als Basis haben. Beispielsweise die Berechnung von Disparitäten oder die verdeckten Pixel könnten bei der Verlustfunktion eines neuronalen Netzerkes exkludieren.

Jede unserer Architekturen bekommt rektifizierte Stereobildpaare als Input. Für das linke und für das rechte Inputstereobild gibt es ein unäres CNN, das die Ähnlichkeit pro Pixel berechnet. Die drei Ansätze unterscheiden sich darin, wie die Outputs der beiden unären CNNs kombiniert werden: Ein Ansatz berechnet die Verdeckungen mit sich erweiternden (dilated) unären CNNs. Die anderen drei Ansätze basieren auf der Stereokorrelation, bei der die zusammengefügten unären CNNs mit erweiterten 2D CNNs oder 3D Faltung verbessert werden.

Wir haben Experimente durchgeführt, in denen wir unsere Ansätze zuerst mit dem synthetischen Datensatz “SceneFlow” (Monkaa) vortrainieren und dann erst mit dem eigentlichen Datensatz “Middlebury” trainieren, wofür wir Adam verwenden. Wir erreichen eine Genauigkeit zwischen 77.98% und 89.02% am “Middlebury” Validierungsdatensatz.

Schlüsselwörter: Verdeckungsdetektierung, Binäre Klassifizierung, Optimierung, Stereokorrelation.

Affidavit

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

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

Eidesstattliche Erklärung

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

Date	Signature

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Patrick Knöbelreiter for the continuous support of my master's thesis study, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor. Independent of any difficulty of any question, he was always able to give me a clear and logical answers, which made my life a lot easier.

Additionally, I would like to thank Thomas Pock for giving me the opportunity to carry out this thesis at this department. I appreciate his engagement for his lectures, that initiated my interest about machine learning.

I want to thank my best friends: Enzo, Günter and Martin, who accompany me for a long period through my life. Thank you fellows for distracting me from university stress and also for your patience when I didn't have time to spend with you. I am so glad that you are part of my life.

Last but not least, I also would like to mention Christian Mentin who is acting as my mentor. He is always ready to listen and give a good advice.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Occlusion in Stereo Vision

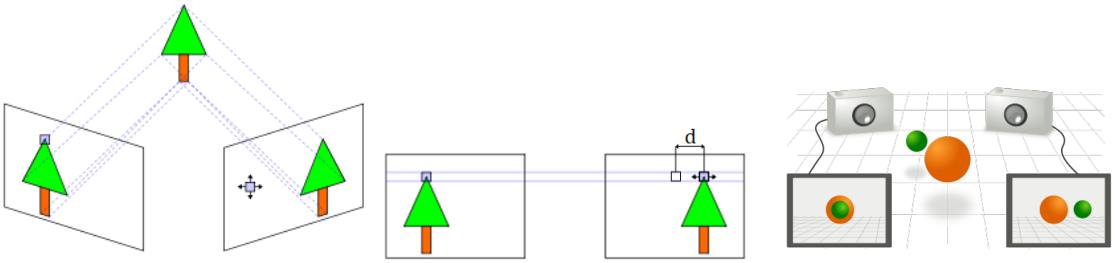
Stereo is a technique for obtaining depth of objects from images captured by two cameras as shown in Figure 1.1a. Stereo vision imitates the ability of the human brain to derive depth from a scene and therefore uses the same principle. The human visual system gathers its stereo information delivered from both eyes. These two views are obtained from two slightly different positions because of the horizontal displacement of our eyes. As a result, a point in the scene of one view is horizontal displaced in the other view. The amount of this displacement can be used to infer the depth.

The amount of the horizontal displacement is called *disparity* (Section 1.1.1). A pixel's disparity is inversely proportional to the pixel's distance from the cameras. With this principle, the human brain transforms the disparity information into a three-dimensional mental image of the world.

Though inferring depth seems to be unproblematic and the stereo vision task is constantly solved by the human visual system without us even noticing the effort, the same task emerges very difficult when it is required to be solved by a computer.

One confronts the major challenge when estimating a disparity map approach is that of solving the correspondence problem. The correspondence problem refers to the task of finding the same pixel in the left and in the right image. This is one of the oldest, but still most challenging problems in the history of low-level computer vision [12].

Some pixels which are not visible in the other view are *occluded*. When we close one eye we can observe that we cannot see some parts of the scene anymore. Figure 1.1c illustrates this problem. The left camera captures an orange and a green ball. The right camera captures a green ball that occludes the orange ball. Hence, the front of the orange ball is not visible in both cameras. The detection of the occluded pixels tells you when you cannot find corresponding point. This would improve algorithms relying on the correspondence problem. In this thesis we investigate machine learning based approaches to detect occluded pixels.



(a) Stereo pair. **(b)** Rectified stereo pair. **(c)** Occlusion.

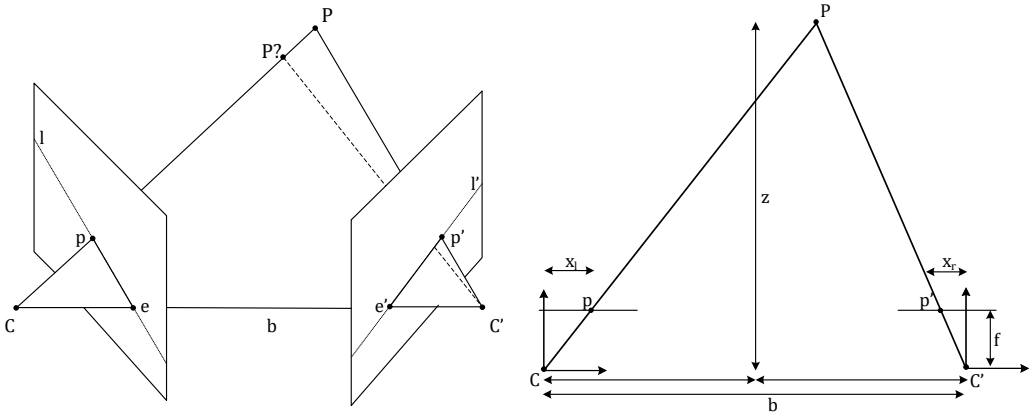
Figure 1.1: Stereo image pair. The left image is fixed and the corresponding pixel is looked up in the right image [1].

1.1.1 Epipolar Geometry and Rectification

If we want to find occluded pixels in stereo images, we have to understand how correspondences are found in order to exploit it for occlusion detection. To simplify the search for correspondences, we can use the observation illustrated in Figure 1.2. Let us assume that we want to find the matching point of a point in the left image $p = [x_l, y_l]^T$ in the right view $p' = [x_r, y_r]^T$. Any scene point $P = [x, y, z]^T$ projected to the image point p is constrained to lie on a line that is the projection ray. As a result, each of those scene points must also lie on a line in the right view. This line is given by the projection of the ray $\overrightarrow{CP}$ into the second image and is referred as epipolar line. The epipoles lie on the epipolar line. These are the points on the image planes which lies at the intersection of the baseline, which is the line connecting the focal centers C and C' with the right image plane. The epipolar line can be determined as $l = e \times p$ and is the result of the cross-product of the epipole e and the point p. It is interesting to note that each epipolar line in the right image must pass through the point e.

The stereo problem is a symmetric problem, the same observations can be made when searching the matching point of p' in the left view. Applying the knowledge about epipolar lines, the correspondence problem can be reduced to a 1D search task. Let us consider the special case if both image planes $\mathbf{I}^L$ and $\mathbf{I}^R$ lie on a common plane and their x-axes are parallel to the baseline. In this setup, the epipoles move to infinity and the epipolar lines coincide with horizontal scan-lines. The matching point of a pixel in one view can

then be found on the same scan-line in the other view (see Figure 1.1b), such that $y_l = y_r$, where y_l and y_r are the y-coordinates of a point in the left and right images, respectively. The horizontal offset between corresponding pixels $x_l - x_r$ is referred to as disparity. To take advantage of this simple geometry, the images of two cameras in general positions can be projected onto a plane that is parallel to the baseline. This process is known as *rectification*. It is more convenient to search for correspondences along horizontal scan-lines rather than to trace general epipolar lines.



(a) Stereo setup with epipolar geometry. **(b)** Triangulation.

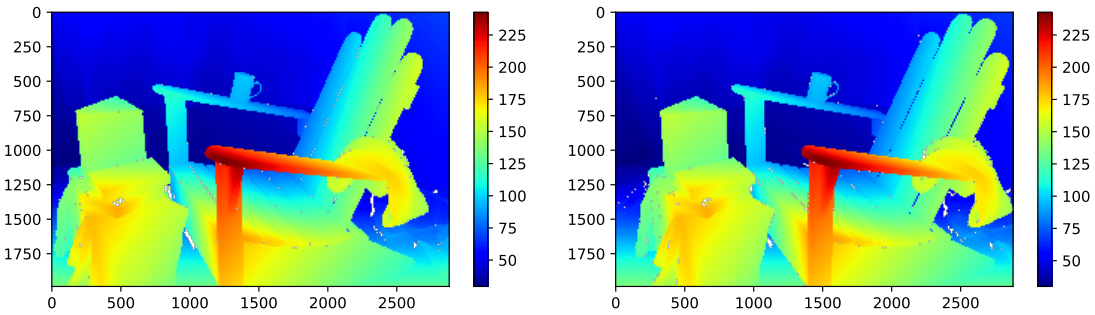
Figure 1.2: Epipolar geometry. P is a point in a scene and p its 2D image on the image plane [2].

1.1.2 Depth Map from Triangulation

Let us assume we know the corresponding points and we assume the cameras are epipolar rectified as described in Section 1.1.1. The reconstruction of a point's depth can then be accomplished via triangulation as shown in Figure 1.2b. This means that the cameras are parallel to each other and have identical focal lengths. The focal length f is the distance between the camera's focal point and its optical center, and the baseline length b. From similar triangles we derive

$$z = \frac{b \cdot f}{x_l - x_r} = \frac{b \cdot f}{d}, \tag{1.1}$$

where b and f are constant and d denotes the disparity. From Equation (1.1) we conclude that disparity is inversely proportional to depth. A disparity map (Figure 1.3) that records the disparity for each image point is therefore sufficient for a complete reconstruction of the scene. This relationship is also the reason why disparity is commonly used as synonym for inverse depth.



(a) Left-to-right disparity map. **(b)** Right-to-left disparity map.

Figure 1.3: Disparity maps [3] of stereo images. The jet color map [4] is used. The color red indicates a high disparity, while black is the lowest disparity.

1.2 Contribution

In this work, we investigate different supervised *CNN* architectures and compare them to each other. The number of methods to obtain occlusions in stereo images is small. Kolmogorov [13] adapted a graph cut algorithm to find correspondences in stereo images. Left-Right-Cross-Checking [14] is the current standard method to handle occlusions. Recently, the first machine learning approach has been published to learn occlusions, called SymmNet [10]. The direct method is similar to SymmNet. Instead of using an hour-glass architecture, we decide to use dilated convolutions. Moreover, we want to avoid using a huge model with a lot of parameters. The remaining architectures are based on stereo correlation [8]. Stereo correlation is a cost volume, where the similarity of potentially corresponding pixels is saved. We design different models, such as dilated convolutions, 3D convolutions, and entropy to exploit the similarity information.

1.3 Notations

This section sums up the notations, which are used throughout this work. Images are functions $\mathbf{I} : \Omega \to \mathbb{R}^c$, where Ω is called the image domain. Ω has the dimension 2 for 2D images. For gray-scale images $c = 1$ and for color images $c = 3$. Every pixel is assigned to a region Ω_i,

$$\Omega = \bigcup_{i=1}^{K} \Omega_i, \qquad \Omega_i \cap \Omega_j = 0, \forall i \neq j,$$

where different regions do not overlap. This work wants to distinguish between occlusions and non-occlusions, so $K = 2$. A classification problem with two classes is called *binary classification*.

We sum up the most common machine learning symbols in Table 1.1. Scalars (e.g. x) are small letters and not bold. In contrast, vectors (e.g. $\boldsymbol{x}$) are bold and column vectors. The k-th element of a vector is accessed via a subscript (e.g. $\boldsymbol{x}_k$). Matrices are uppercase letters with a bold face (e.g. $\mathbf{M}$) and the elements are row-major ordered. To access an element in row i and column j we denote it as $\mathbf{M}_{i,j}$. Tensors are italic uppercase letters with a bold face (e.g. $\boldsymbol{T}$). It has one dimension more. To access an element in row i, column j and depth k, we denote it as $\boldsymbol{T}_{i,j,k}$.

Sets are denoted by uppercase calligraphic letters such as $\mathcal{X}$ or $\mathcal{Y}$. The number of elements within a set can be expressed as the cardinality of a set $|\mathcal{X}|$.

Let $(\mathcal{X},\ \mathcal{Y})$ be the data-set and its class labels, where $\mathcal{X}$ is a d-dimensional feature space $\mathbb{R}$. Samples of the data-set $\boldsymbol{x} \in \mathcal{X}$ can be drawn. Each entry in $\boldsymbol{x}_i$ represents an attribute and is called feature. The samples are distinguished by using a superscript, $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}\}$. A subset of labels is denoted as $\boldsymbol{x}^{(1:m)}$, where $m < N$. The class labels are natural numbers $\mathcal{Y} = \{1, ..., Y\}$. The cardinality of the class labels, $|\mathcal{Y}| = 2$, means that we only have 2 classes and hence a binary classification problem. To each labelled data point $\boldsymbol{x}$, a label $y \in \{0, 1\}$ is assigned. The predicted values or the output of the method is denoted as $\hat{y}$.

In Table 1.2, the hyper-parameters are listed. In machine learning, hyper-parameters are the parameters whose values are set before starting the learning process. The most prominent letter is the learning rate η and is responsible for the step size of a loss function.

Symbols	Meaning
x	Scalar
$\mathbf{x}$	Column vector
$\boldsymbol{x}_k$	The k-th element of the vector $\boldsymbol{x}$
$\mathbf{M}$	Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$
$\mathbf{M}_{i,j}$	The element in the i-th row and j-th column of the Matrix, $i < m$, $j < n$
$\mathbf{M}^T$	Matrix transposed
$\boldsymbol{T}$	Tensor $\boldsymbol{T} \in \mathbb{R}^{m \times n \times k}$
$\boldsymbol{T}_{i,j,k}$	The element in the i-th row, j-th column and k-th depth
$y \in \mathcal{Y}$	Labels
$(X, Y) \in \mathcal{X}$	Training-set with tuples of input x and its label y
θ	Parameters, that are learned weights and bias, e.g. $\theta = \{\mathbf{W}, b\}$
ϕ	Feature
$\mathcal{L}$	Loss
$\sigma(x)$	Logistic sigmoid, i.e.$(1 + \exp(-x))^{-1}$
σ^2	Variance
σ	Standard deviation
μ	Mean
$\mathcal{N}(\mu, \mathbf{\Sigma})$	The Gaussian distribution with mean μ and co-variance matrix $\mathbf{\Sigma}$

Table 1.1: Common machine learning notations.

Symbols	Meaning
η	Learning rate, e.g. $1e^{-4}$
β_1	Parameter of the first momentum, i.e. 0.9
β_2	Parameter of the second momentum, i.e. 0.99
λ, γ	Penalty

Table 1.2: Hyper-parameters.

1.4 Outline

In this section, we give a brief overview of the main points of each chapter. It should clarify the structure of this thesis to quickly find the relevant chapters.

Chapter 2 In the **Related Work** we discuss basic concepts of stereo matching to state-of-the-art methods for stereo matching and occlusions detection. Both topics stereo matching and occlusions are closely related to each other as occluded pixels cannot be matched.

Machine learning is the underlying method in this work and so we want to explain the mathematical concept of a Convolutional Neural Network (*CNN*). Therefore, a detailed explanation of *CNN*s is given. It is necessary to understand this concept for the stereo correlation that is explained in Section 3.2.

Chapter 3 Chapter **Methods** represent different methods that we investigate throughout this work. In general, we have two groups of methods: The first group learns occlusions directly, without prior-knowledge. The second group calculates a disparity volume first and based on it three methods calculate the occlusion map.

Chapter 4 The **Evaluation** contains the metric setup to measure the performance and the experiments. As a metric, we use the Receiver Operating Characteristic curve to find a threshold to distinguish between occluded and non-occluded pixels. Finally, we report the quantitative and qualitative results of our work. We compare different models to each other.

Chapter 5 The **Conclusion** concludes this work by discussing the outcome of the methods. This includes weaknesses of the methods' architectures, as well as giving an outlook on future work.

2 Related Work

Contents

Important publications related to this thesis are discussed in this chapter. The first section focus on simple methods which have already been used in the early stage of stereo matching. One of the more intuitive ways to compare pixel intensities is to use either the $L1$ or the $L2$-norm to compare corresponding pixels. These pixels which gives the lowest distance are most similar. One problem of these methods is if there are homogeneous regions and the pixel intensities are very similar to each other. One improvement is to take the neighboring pixels into account. In literature, these methods are summed up as *Block Matching* methods [15], where small region around a certain point is compared to another region from another point. We will explain roughly CT and Sum of Squared Differences (SSD), but there are many other block matching approaches, such as Sum-of-Absolute-Differences (SAD) or Normalized Cross-Correlation (NCC).

Then, the next section explains the concept of neural networks. Basic concepts like neurons, different layers and common activation functions are explained. Especially those layers of Convolutional Neural Network (CNN) are presented, which have shown their importance in image processing in the recent years. We also discuss different loss functions and optimizer to train a CNN. During training a CNN several problems can occur such as over- or under-fitting. We will describe how to counteract such problems.

Furthermore, we provide an overview of machine learning methods to find point correspondences in stereo image pairs. People tried to combine former methods with machine learning. We discuss one approach that uses post-processing methods like Semi Global Matching (SGM) to refine their results. We also treat another approach which learns a disparity map without post-processing techniques. We will utilize this method to detect occlusion, but this described in detail in the Chapter 3 "Methods".

In the last section, we give an overview of three state-of-the-art occlusion detection methods: Left-Right-Cross-Checking (LRC) method, graph cut method and a machine learning method named "SymmNet". The purpose to discuss various methods is to not only deal with machine learning methods in this thesis. These methods are often used for stereo matching purposes and are a good example for utilizing these methods to occlusion detection.

2.1 Block Matching Approaches

Block matching approaches have together that they compare a pixel with a pixel at an another position, if it is the same pixel. The comparison of the pure color intensities of one pixel is not enough because .there could be a lot more pixels with the same color intensities. If we take the neighboring pixels into account, it usually depicts more successful results. The pixel to compare and it's neighbors are lying under the block. The block on the reference image moves on and the comparison results are saved. For occlusion detection is the block that matches least a possible candidate.

2.1.1 Census Transform

Census Transform (CT) [16] is a stereo correspondences algorithm that relies on the pixels gray-scale intensity values. A drawback of the CT algorithm is its incorrect matches [17] in regions of repetitive structures. A pixel is compared to its local neighbors in a 3×3 window of the left $\mathbf{P}^L$ and right $\mathbf{P}^R$ stereo image pair respectively.

A census is the procedure of recording pixel values that are assigned to local neighborhood. So a 8-bit mask records the local neighborhood relation. If the intensity value of the neighboring pixel is smaller than the intensity of the center pixel, then 1 is appended to the mask. Otherwise, 0 is appended to the mask. Finally, we have a bit-mask for both images.

$$\mathbf{P}^L = \begin{bmatrix} 10 & 12 & 28 \\ 20 & \mathbf{30} & 42 \\ 80 & 82 & 84 \end{bmatrix} \quad CT(\mathbf{P}^L) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & C & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{P}^R = \begin{bmatrix} 11 & 15 & 34 \\ 35 & \mathbf{30} & 45 \\ 82 & 85 & 85 \end{bmatrix} \quad CT(\mathbf{P}^R) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & C & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Figure 2.1: Example of 3×3 image pair where the Census Transform (CT) is applied on the centering pixel in bold.

In Figure 2.1, we have exemplary two regions from the left and right images. The CT is calculated for both regions so that the 8-bit masks are $CT(\mathbf{P}^L) = 00\mathbf{00}111\mathbf{1}1$ and $CT(\mathbf{P}^R) = 00\mathbf{111}0\mathbf{1}1$. Then, the Hamming distance can be calculated of both bit masks. This distance counts the number of the bits which differ. In our case the Hamming distance is 3. A high Hamming distance denotes that the image regions are not very similar, whereas a low Hamming distance denotes that centering pixels are corresponding. The pixel in the right image with lowest hamming distance is chosen as the corresponding pixel in the left image.

Meister et al. [18] extended the CT to handle occlusions. They use the former 0 to mark occlusions and add -1 for mismatching intensity values, i.e. the census of the left image patch is $CT(\mathbf{P}^L)$ =-1-1-111-11010. The occlusions needs to be calculated from a pre-computed occlusion map.

2.1.2 Sum of Squared Differences

Sum of Squared Differences (*SSD*) compares the intensity values of the left and right image. As we assume that the images are rectified, we only look for most similar pixel in the right image in the same row. The mere intensity values are compared, where the corresponding pixel with smallest distance is chosen as the most similar pixel. The *SSD* for correspondences problems is defined as follows:

$$\mathrm{SSD}_{i,j} = \sum_d (\mathbf{I}^L_{i,j} - \mathbf{I}^R_{i,j-d})^2 = \sum_d (\mathbf{I}^L_{i,j})^2 + (\mathbf{I}^R_{i,j-d})^2 - 2\underbrace{\mathbf{I}^L_{i,j}\mathbf{I}^R_{i,j-d}}_{\text{correlation}}, \tag{2.1}$$

where the disparity d changes the position in the row in the right image, so that the most similar pixel corresponding to the pixel in the left image can be found. The term which is under-braced with "correlation" computes the similarity pixel-wise. The correlation term is subtracted from the other squared terms and the smaller the result are, the more similar the corresponding pixels, where occluded pixels can also be considered.

If we expand the Equation (2.1) to block-matching, we need to introduce m and n. The indices m and n define the window around a pixel. The pixel intensities within this window in the left image are compared to the pixel intensities in the right image. A smaller window size allows to compute more details. A larger window size counteracts few isolated mistakes, where it will be damped that occluded are matched. The *SSD* with blocks is defined as

$$\mathrm{SSD}_{i,j} = \sum_{m,n} (\mathbf{I}^L_{i,j} - \mathbf{I}^R_{i-m,j-n})^2 = \sum_{m,n} (\mathbf{I}^L_{i,j})^2 + (\mathbf{I}^R_{i-m,j-n})^2 - 2\underbrace{\mathbf{I}^L_{i,j}\mathbf{I}^R_{i-m,j-n}}_{\text{correlation}}, \tag{2.2}$$

The Equation (2.2) yields the (cross-)correlation (compare to Equation (2.5)). The correlation is explained in detail in Section 2.2.3.2, but for now, it is enough to know that Convolutional Neural Networks also calculates the correlation.

2.2 Principle of a Feed-forward Neural Network

Neural networks are biologically inspired by Hubel and Wiesel's early work [19] on the functional instigation of a cat's cortex. The visual cortex cells act as local kernels and use spatially local correlation present in natural images. A neural network is a mathematical model to emulate this natural behavior. It is a directed acyclic graph. This kind of networks is called *feed-forward* neural networks. In other words, there are no loops in the network as in recurrent neural networks.

This section introduces the neural network model. First, the basic concept is explained from a neuron, convolutional kernels, to layers. The second step is how to train these networks with optimization methods.

2.2.1 Neuron

Neural Networks consists of artificial neurons, where there are different kind of neurons such as the perceptrons[1] [20] (see Figure 2.2) for a binary classifier. A perceptron is a simpler form of a neuron. It gets three inputs, e.g. $x_1,\ x_2,\ x_3 \in \{0, 1\}$, that can be expressed as binary encoding. The input x_1 can be 1 and the others can be 0. Each input is weighted by $w_1,\ w_2,\ w_3 \in \mathbb{R}$, that express the importance of the input.

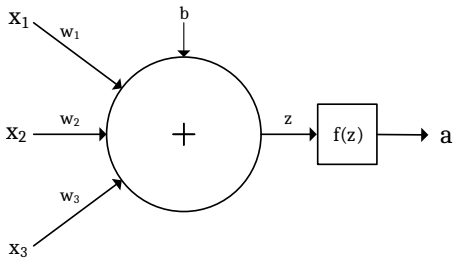


Figure 2.2: Neuron with 3 inputs ($x_1,\ x_2,\ x_3$).

The output of a perceptron is expressed by the weighted sum $\sum_j \boldsymbol{x}_j \boldsymbol{w}_j = \boldsymbol{x}^T \cdot \boldsymbol{w}$ which is less or greater than a certain threshold value, the bias b, that is a real number.

$$z = \begin{cases} 0 & \text{if } \boldsymbol{x}^T \cdot \boldsymbol{w} + b \leq 0, \\ 1 & \text{else.} \end{cases}$$

The bias b is a parameter of the perceptron and helps to find a better decision boundary to separate data. The perceptron is 0 if less than or equal to b, or 1 if greater than b.

A neuron differs from the perceptron by the activation function. There is usually a non-linearity ($f = \{\tanh, \text{sigmoid}, \text{ReLU}, \text{leaky ReLU}, \dots\}$) warped around a perceptron.

[1] The idea of a perceptron was inspired by a biological neuron as humans have in their nerve cells.

The activation a of a neuron can be written as

$$a = f(\boldsymbol{x}^T \cdot \boldsymbol{w} + b) \tag{2.3}$$

with f being its activation function that decides when a neuron is activated. Only non-linear activation functions allow a neural network to compute nontrivial problems by using a small number of nodes.

2.2.2 Activation Functions

Activation functions $f : \mathbb{R} \to \mathbb{R}$ define the output of one neuron in Equation (2.3). This output is used as input for the next neuron and so on. If there would not be any non-linearity [21] the whole neural network would act as one linear neuron. A linear output can only solve linear separable problems and thus mapping to non-linear space is necessary.

The Universal Approximator Theorem (UAT) [22] claims that a feed-forward network with one layer that contains a finite number of neurons and an arbitrary activation function are universal approximators. Suppose we have given a function $f(x)$ which the neural network would like to approximate within some accuracy $\varepsilon > 0$. If we use enough neurons the neural network $g(x)$ will satisfy

$$|g(x) - f(x)| < \varepsilon$$

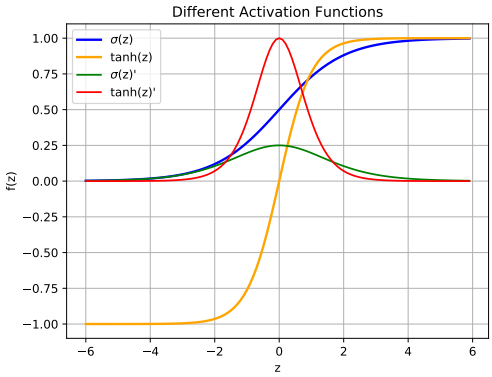
for all inputs x. It is important to emphasize that the UAT guarantees that there is a solution, but not how good a function can be approximated. The choice of an activation function f has an impact on the quality of the solution. The importance of activation functions in neural networks has been proposed for different applications because the right activation function can lead to a performance increase. Activation functions can typically be separated into two families according to the curve shape i.e. *sigmoidal* (sigmoid, tanh) and *non-sigmoidal* (ReLU, lReLU).

Sigmoid It has the characteristic s-shape and squashes the input values in a finite boundary $[0, 1]$. It is also continuous and differential for all real-valued inputs. The sigmoid function is defined as follow:

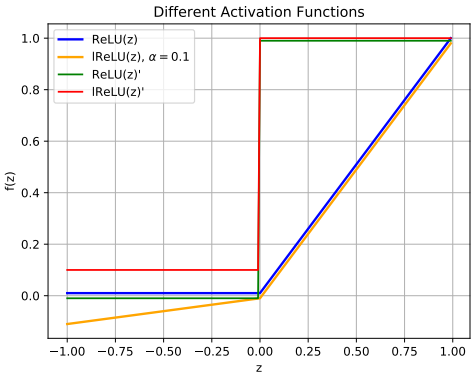
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and its derivative w.r.t. to z:

$$\begin{aligned}
\frac{\partial}{\partial z}\sigma(z) &= \frac{\partial}{\partial z}\frac{1}{1+e^{-z}} = \frac{\partial}{\partial z}(1+e^{-z})^{-1} = \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{(1+e^{-z})-1}{(1+e^{-z})(1+e^{-z})} = \frac{1}{1+e^{-z}}\Big(1 - \frac{1}{1+e^{-z}}\Big) \\
&= \sigma(z)\big(1-\sigma(z)\big).
\end{aligned}$$



(a) Sigmoid and tanh with their derivatives.



(b) ReLU and leaky ReLU with their derivatives.

Figure 2.3: Different activation functions and their derivatives.

Hyperbolic Tangent - tanh It has a wider output range $[-1, 1]$ compared to the sigmoid function and the curve midpoint is at origin of the coordinate system. The tanh is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z - e^{-z}}.$$

The derivation of $\tanh(z)$ w.r.t. to z:

$$\begin{aligned}\frac{\partial}{\partial z}\tanh(z) &= \frac{\partial}{\partial z}\frac{e^z - e^{-z}}{e^z - e^{-z}} = \frac{(e^z + e^{-z})(e^z + e^{-z}) - (e^z - e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2} \\ &= 1 - \frac{(e^z - e^{-z})^2}{(e^z + e^{-z})^2} = 1 - \tanh(z)^2.\end{aligned}$$

Rectified Linear Unit - ReLU This activation function [23] is in general compute efficient, since it needs a simple comparison of two values and does not use any exponential function as the sigmoidal functions. For negative input values ReLU produces zero-gradient, see Figure 2.3b:

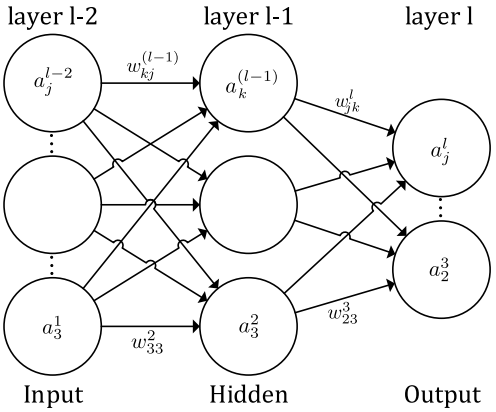
$$\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0, \end{cases} \qquad \frac{\partial}{\partial z}\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ [0, 1] & \text{if } z = 0 \\ 1 & \text{if } z > 0. \end{cases}$$

Therefore, a small skew $\alpha \in (0, 1]$ (typically $\alpha = 0.1$) is introduced and the lReLU [24] does not have zero partial derivatives:

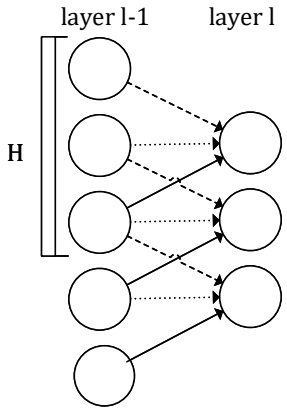
$$\text{lReLU}(z) = \begin{cases} \alpha z & \text{if } z < 0 \\ z & \text{if } z \geq 0, \end{cases} \qquad \frac{\partial}{\partial z}\text{lReLU}(z) = \begin{cases} \alpha & \text{if } z < 0 \\ [\alpha, 1] & \text{if } z = 0 \\ 1 & \text{if } z > 0. \end{cases}$$

2.2.3 Layer

If we want to solve more complicated problems, we need to put more neurons together. This forms a layer and has the property that whether the input nor the output the neurons are exchanged within one layer. Only the output of the neurons in the previous layer is taken as input in the current layer. The output of the current layer can only taken as in put for the next layer. It depends on the architecture which neurons are connected from following layers. In this section, we treat fully-connected and convolutional layers.



(a) Multilayer Perceptron (*MLP*) [25].



(b) Sectional view of a *CNN*.

Figure 2.4: Comparison of the *MLP* with a 2D convolutional network with the kernel **H** (see Section 2.2.3.2).

2.2.3.1 Fully-Connected Multi-Layer Perceptron

A Multilayer Perceptron (*MLP*) [25] is split into 3 parts: input, hidden and an output layer. The input layer can be pixels of images or speech recordings. The hidden layer(s) follow(s) after the input layer and does not count to the output layer. A perceptron in a Multilayer Perceptron (*MLP*) is connected to all perceptrons in the following layer (fully-connected, Figure 2.4a). If we want to classify numbers as in MNIST dataset [26], we have as input 64×64 grey-scaled images from digits 0,...,9. We would need 4096 perceptrons for this image size. The output layer, which is a single perceptron in our case, indicates if

the input image 9 by less than 0.5 and values greater than 0.5 indicating the input image classified to the digit "9".

2.2.3.2 2D Convolutional Layer

A Convolutional Neural Network (*CNN*), invented by LeCunn et al. [27], introduces besides weights and bias additional a kernel for weight sharing. Several neurons use the same weights under a kernel **H**. The weight sharing principle is shown in Figure 2.4b as section view, where different dashed lines give a showcase of same weight occurrence. In addition, not every neuron is connected to a neuron in the following layer. We have to distinguish between parameters (weights and bias), which are learned during training and hyper-parameters of the model whose values are set before the learning process starts [28].

Hyper-parameters We have the common hyper-parameters listed in the Table 1.2. Each layer has specific hyper-parameters, which specify the number of connections and the output size of the feature maps.

Kernel Size The kernel size that is usually smaller than the input. If the size of the kernel is increased, it allows to process more spatial information. At the same time the receptive field (see Figure 2.6) is increased as well.

Stride A convolution is a weighted summation by sliding a kernel over an input. Striding specifies how much a kernel is moved between every feature computation. However, a stride of 1 moves the kernel once at a time and does not influence the feature map output size. Larger strides have smaller output feature maps [28]. It is often used to downsample images such as in hour-glass architectures [10].

$$\mathbf{U} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \mathbf{6} \end{bmatrix} \qquad \mathbf{U}_z = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{2} & \mathbf{3} & 0 \\ 0 & \mathbf{4} & \mathbf{5} & \mathbf{6} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{U}_s = \begin{bmatrix} 2 & 1 & 1 & 2 & 3 & 3 & 2 \\ 2 & 1 & \mathbf{1} & \mathbf{2} & \mathbf{3} & 3 & 2 \\ 5 & 4 & \mathbf{4} & \mathbf{5} & \mathbf{6} & 6 & 5 \\ 5 & 4 & 4 & 5 & 6 & 6 & 5 \end{bmatrix}$$

Figure 2.5: $\mathbf{U}$ - no padding. $\mathbf{U}_z$ - zero padding. $\mathbf{U}_s$ - symmetric padding.

Padding On the basis of the definition of the convolution operation in Equation (2.4), the convolution shrinks the input, because the kernel is strictly inside the image. Due to the summation to the centered pixel, the pixels closer to the margin cannot be kept. It is convenient to keep the spatial dimensions of the input. At this point padding can be applied, where the input $\mathbf{U}$ is extended with numerical values at the borders. These values can have an impact on the quality of the output at the border

(Figure 2.5). There are two types of padding For example, if we would apply a 3×3 kernel to input $\mathbf{U}_z$ with zero-padding (see Figure 2.5), after the 2D convolution the result would have dimensions of $\mathbf{U}$. Therefore, for several layers, we would need to pad more zeros at the border. For the input $\mathbf{U}_s$ with smmetric padding we could apply a kernel with the size of 3×5. This time $\mathbf{U}$ is extend with a symmetric padding [29], that helps to not have veils at the borders on the final output. In addition it is convenient to distinguish between *same* and *valid* padding. The machine learning framework tensorflow [29] adds exactly as many of zeros around the input $\mathbf{U}$ so that the same size of $\mathbf{U}$ is kept. Valid padding must not be zeros added around the input. There different paddings such as "constant", "mirror" or "symmetric". After 2D convolution on a valid padded input, the output dimension must not be necessarily same size as the input dimension.

Number of learnable Parameters The number of parameters directly corresponds to the size of kernel and biases. Each kernel produces an output feature map, hence k kernels produces an output feature map of depth k [28]. If we neglect striding, the total number of learnable parameters is the sum of all weights and biases of all layers. Consider a kernel size of $n \times m$, input features n_{in} plus a bias of each feature map, which maps to the output features n_{out}. The number of parameters n_p can be calculated as

$$n_p = (n \cdot m \cdot n_{in} + 1) \cdot n_{out}.$$

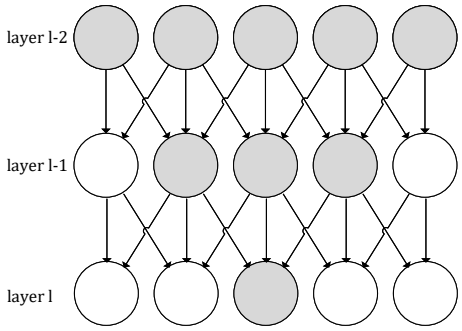


Figure 2.6: Receptive field (grey) over several layers (l is the current layer).

Receptive Field When we look at one input space, we can state where a feature is looking at. Unlike for fully connected neural networks, where the value of a unit depends on the entire input [30]. In contrast, a unit in a *CNN* depends on a region of input. This region in the input is called the receptive field for that one unit. This input region can also be the output from other units in the neural network. To increase the receptive field different methods can be used such as sub-sampling (pooling, striding) of the network, dilated convolutions [31] or simply add more layers to increase the depth, where the receptive field is increased linearly in theory.

2D Convolution and Correlation In current popular deep learning libraries, such as Tensorflow [29], the naming convention 2D convolution is often used, but actually a correlation is performed. Basically, only the kernel is flipped (rotated by 180 degrees, Equation (2.6)). The output at pixel position (i, j) of the convolution can be calculated by dot products between the entries of the kernel $\mathbf{H}$ with the input $\mathbf{U}$:

$$(\mathbf{U} * \mathbf{H})_{i,j} = \sum_{m,n} \mathbf{U}_{i-m,j-n} \cdot \mathbf{H}_{m,n}. \tag{2.4}$$

Analogous, the 2D correlation can be written as

$$(\mathbf{U} \otimes \mathbf{H})_{i,j} = \sum_{m,n} \mathbf{U}_{i+m,j+n} \cdot \mathbf{H}_{m,n}. \tag{2.5}$$

As long as the kernel is symmetric, there is no difference between correlation and convolution. Equation (2.5) is the fundamental equation for the 2D correlation layer, where $\mathbf{U}$ is the image and each neuron corresponds to a pixel. The 180-degree rotation is applied to the kernel $\mathbf{H}$:

$$(\mathbf{U} * \mathrm{r}_{180}(\mathbf{H})) = \left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} h_9 & h_8 & h_7 \\ h_6 & h_5 & h_4 \\ h_3 & h_2 & h_1 \end{bmatrix} \right). \tag{2.6}$$

2.2.3.3 2D Dilated Convolution Layer

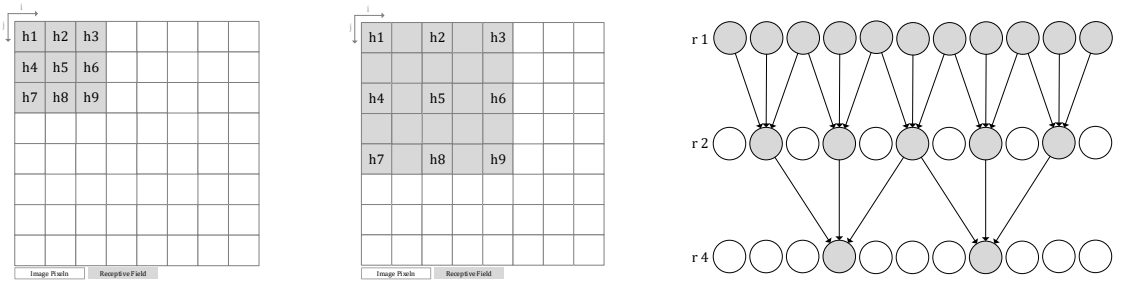
This dilated/atrous convolution [32] is a special case of the 2D convolution in Section 2.2.3.2. A parameter rate r is introduced

$$(\mathbf{U} *_r \mathbf{H})_{i,j} = \sum_{i,j} \mathbf{U}_{(i+r)-m,(j+r)-n} \cdot \mathbf{H}_{m,n} \tag{2.7}$$

that produces "holes" (in French: atrous) between the pixels of the input image. Dilated convolutions expand the receptive field exponentially without losing resolution. In simple terms, dilated convolution is just a convolution applied to input with defined gaps. The grey area is the current position of the kernel on the input. The Figure 2.7 shows dilated convolution on 2D data. A 3×3 kernel $\mathbf{H}$ is altered by a dilated convolution. A rate of 1 (Figure 2.7a) does not change the kernel size. A rate of 2 (Figure 2.7b) does increase the receptive field size to 5×5. With these definitions, given our input is a 2D image, dilation rate $r = 1$ is a normal convolution and $r = 2$ means skipping one pixel per input and $r = 4$ means skipping 3 pixels as in Figure 2.7c.

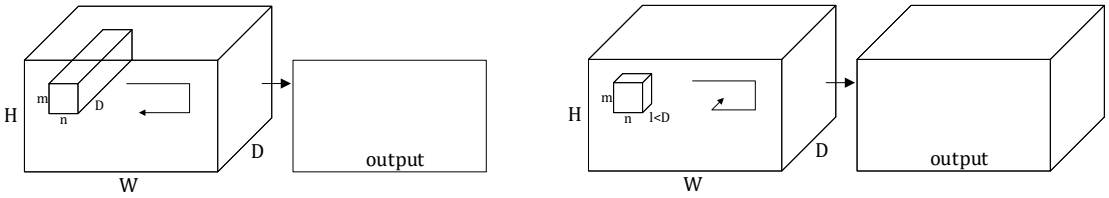
The grey area is also the receptive field and depends on the size of the kernel. The receptive field of one neuron increases per layer (see Figure 2.7c). Systematic dilation increase supports an exponential expansion of the receptive field without loss of resolution

or coverage, while the number of parameters grows only linearly.



(a) Dilation rate 1. **(b)** Dilation rate 2. **(c)** Dilated convolutions with rate $r = \{1, 2, 4\}$.

Figure 2.7: Dilated convolution with different rates and over several layers.



(a) 2D convolution. **(b)** 3D convolution.

Figure 2.8: [5] illustrated the 2D and 3D convolution operations.

2.2.3.4 3D Convolution Layer

3D convolution give promising results for spatial learning problems [5, 33]. When we train our network for a 2D convolution in Equation (2.4), it sums up over the whole volume. When we use a 3D convolution in Equation (2.8), we can consider some different disparities and the 2D convolution is extended with a 3$^{\text{rd}}$ dimension

$$(\boldsymbol{U} * \boldsymbol{H})_{i,j,k} = \sum_{m,n,l} \boldsymbol{U}_{i-m,j-n,k-l} \cdot \boldsymbol{H}_{m,n,l}, \tag{2.8}$$

where k is the new index. Compared to the 2D convolution Equation (2.4) which produces a new image, the 3D convolution Equation (2.8) produces a new output volume. When we would choose D that is exactly the depth of the volume as in Figure 2.8, then the 3D convolution works identically as the 2D convolution.

2.3 Training a Neural Network

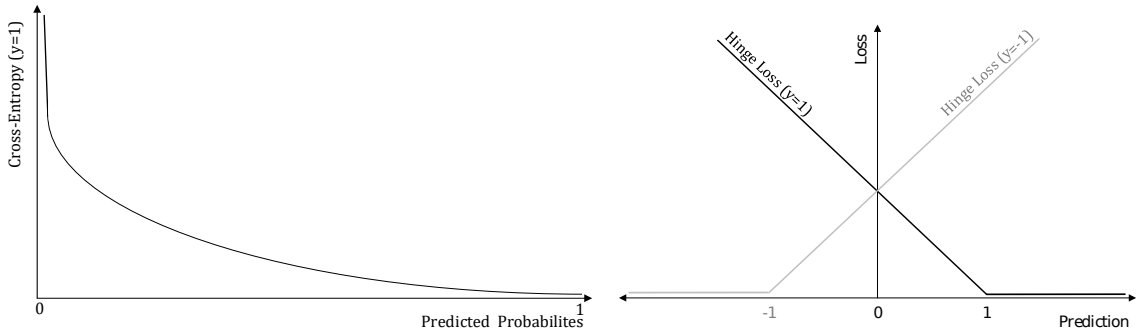
At this point we know the basic building blocks of a neural network. We have to train it according to a training set of size N. A neural network "learns" by solving an optimization problem to choose a set of parameters that minimize a certain loss function $\mathcal{L}$ in order to reduce costs. Weights will be initialized by random values (see Section 2.3.5). As in most neural networks the error is calculated

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2} ||y - \hat{y}||^2, \qquad x \in \mathbb{R}^N,\ y \in \{0, 1\},\ \hat{y} \in [0, 1],$$

by the difference between the prediction $\hat{y}$ (the activation of the last layer $\boldsymbol{a}^L$) and the ground truth $y \in \mathcal{Y}$, the actual output. The function that is used to measure this error is called loss function $\mathcal{L}$.

2.3.1 Loss Functions

Loss functions measure the error between prediction and ground truth. Different loss functions measure a different error for the same prediction. Hence, loss functions have a considerable impact on the performance of the model, but it also depends on the task for what the model should be trained for, e.g. classification or regression. Both Cross-Entropy (CE) and Hinge Loss (HL) can be used for binary classification and are therefore suitable to classify pixel-wise whether it is occluded or not.



(a) Cross-Entropy Loss. **(b)** Hinge Loss.

Figure 2.9: Different loss functions.

2.3.1.1 Cross-Entropy

For our learning problem with a training set size of $|\mathcal{X}| = N$, we decided to take the Cross-Entropy (CE) as loss function

$$CE(y, \hat{y}) = -\frac{1}{N} \sum_{i}^{N} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

to compare in general two probability distributions. For one class the adds $\log \hat{y}^{(i)}$ to the loss and for the other class the *CE* adds $\log(1 - \hat{y}^{(i)})$. When the neuron's actual output is close to actual output for all training input, the *CE* also goes to zero (see Figure 2.9). The loss is high when the neuron's actual output is far away to the actual output. When we derivate the *CE* w.r.t. w

$$\begin{aligned}\frac{\partial CE}{\partial w} &= -\frac{1}{N}\sum_x \left(\frac{t}{\sigma(z)} - \frac{(1-y)}{1-\sigma(x)}\right)\frac{\partial \sigma}{\partial w} = -\frac{1}{N}\sum_x \left(\frac{t}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x \\ &= -\frac{1}{N}\sum_x x(\sigma(z) - y),\end{aligned}$$

it provides us the rate that a weight learns is controlled by $\sigma(z) - y$, it is the distance or error in the output [25]. Intuitively, the larger the error the faster the neuron learns. The $\sigma(z)'$ is canceled out and this avoids the learning slowdown.

2.3.1.2 Hinge Loss

Hinge Loss (*HL*) is usually used for "maximum-margin" classification, particularly for a Support Vector Machine (SVM). A SVM is a supervised learning model to analyze data for classification or regression. A hyper-plane is found which separates best the training data. The *HL* function is not differentiable, but has sub-gradients w.r.t. w. It is defined as

$$HL(y, \hat{y}) = \max(0, 1 - y\hat{y}),$$

where $y \in \{-1, 1\}$ and $\hat{y}$ is the raw prediction of the classifier. The *HL* does not only penalize incorrect predictions as in Figure 2.9a, it penalize correct predictions as well when they are not confident enough. The gradient of the *HL* can be calculated via sub-gradients:

$$\begin{aligned}\frac{\partial HL}{\partial w} &= \frac{\partial HL}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} \\ &= \begin{cases} 0 & \text{if } yf(w^l a^{l-1} + b^l) > 1, \\ [-ya^{l-1}, 0] & f(w^l a^{l-1} + b^l) = 0, \\ -ya^{l-1} & \text{else,} \end{cases}\end{aligned}$$

where a^{l-1} (Equation (2.3)) is the output of the previous layer. The incorrect predictions are penalized linearly.

2.3.2 Minimize a Loss Function

The goal of optimizing *CNN*s is to find weights and biases such that the network approximates the desired output $\hat{\boldsymbol{y}}$ to a given input $\boldsymbol{x}$. We need to define a measurement for how well the *CNN* approximates the output. This measurement is referred to as cost function

$J(\theta)$, where θ stands for the network parameters (weights, bias). The initialization of these network parameters θ gives a good start position. We will explain some initialization heuristics in Section 2.3.5. We have N training examples, so that a data set is given $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}\}$ and the corresponding labels by $\mathcal{Y} = \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, ..., \boldsymbol{y}^{(N)}\}$. $J(\theta)$ is computed as the average over every sample loss function $\mathcal{L}(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \theta))$ [28]:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \theta)).$$

2.3.2.1 (Stochastic) Gradient Descent

The Gradient Descent (GD) algorithm requires a lot of computational resources when the training set is very large because one update step requires the computation of all gradients of all training examples as in Algorithm 1. In contrast, the Stochastic Gradient Descent (*SGD*) accelerates the learning process by calculating the gradient of the cost function $\nabla_\theta J(\theta)$ by computing the gradients on a small subset of m randomly chosen training examples $\mathcal{X}_m$ from the training set $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}...., \boldsymbol{x}^{(N)}\}$ and the corresponding label $\mathcal{Y}_m$ from $\mathcal{Y} = \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, ..., \boldsymbol{y}^{(N)}\}$. The subsets $\mathcal{X}_m$ and $\mathcal{Y}_m$ are called mini-batch with the mini-batch size m [25]. If the mini-batch size is very small, only a very rough approximation of the actual gradient can be achieved, but in practice, *SGD* has been shown to converge well [28]. *SGD* is probably one of the most used optimization algorithms for neural networks.

Algorithm 1: Gradient Descent.

Data: $\theta = \{\mathbf{W}, \boldsymbol{b}\}$.
Result: Optimal θ.
repeat
  $\theta_{t+1} = \theta_t - \eta \nabla_t J(\theta_t; \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)})$
until *convergence*;

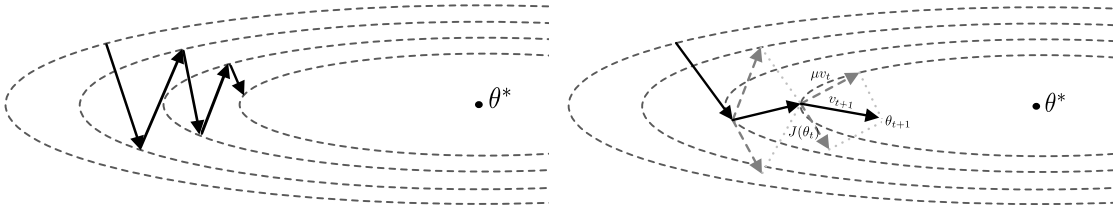
Algorithm 2: Gradient Descent with momentum.

Data: $\theta = \{\mathbf{W}, \boldsymbol{b}\}$.
Result: Optimal θ.
repeat
  $v = \mu v - \eta \nabla_{\theta_t} J(\theta_t; \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)})$
  $\theta_{t+1} = \theta_t + v$
until *convergence*;

2.3.2.2 Momentum

A problem of the GD and *SGD* is that the convergence can be very slow since the gradient steps become smaller when we approach a local minimum, reaching the bottom takes a long time. It can happen that the GD stagnate in local minima or saddle points. The

momentum helps to damp oscillations and to go through narrow valleys as well as local minima [34]. The idea of the momentum comes from a physical model, the heavy ball method [34]. A ball rolls down a slope and is accelerated by its own mass. Momentum is often called *acceleration* in literature[2]. Thus, the momentum can accumulate velocity across epochs in the direction where the gradient points towards. Mathematically, a zero-initialized variable v is introduced that is added [35] to parameters θ_t (see Algorithm 2) at each iteration. The hyper-parameter μ (typically $\mu = 0.9$) reduces the velocity, so that the GD can stop at the bottom of the slope.



(a) Gradient descent GD. **(b)** GD with momentum.

Figure 2.10: Gradient (black arrows) move at each epoch towards global minima θ^*.

The Figure 2.10 illustrates GD and GD with momentum. The dotted ellipses are the level lines of the loss surface. The point θ^* represents the global minimum, where the loss is equal to zero. It can be observed that GD with momentum is closer to the global minima as the GD without momentum.

2.3.2.3 Optimizer with Adaptive Learning Rate

The adaptive learning rate is a crucial instrument because the direction of the gradient is very sensitive. The moment algorithm can soften this issue, but it introduces new hyper-parameters. As we think that the directions of sensitivity are somewhat axis-related, it does make sense to use separate learning rate for each parameter and adapt these learning rates during training. The delta-bar-delta algorithm [36] is an early adaptive approach based on a simple idea: If the partial derivative w.r.t. the given parameter, remains the same sign, then the learning rate should be increased. If it changes the sign, the learning rate should be decreased.

In this section, we want to summarize briefly different methods to adapt the learning rate optimizer, where earlier optimizer establish a foundation for later optimizer. Therefore, we list the adaptive optimizer in chronological order.

AdaGrad AdaGrad is the abbreviation for *adaptive gradient* [37] and is at the same time the first of the introduced algorithms. The main idea of this optimizer is to keep track

[2] `https://distill.pub/2017/momentum`

of the sum of squared gradients for each parameter. Basically, it is a modified stochastic gradient descent

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbf{G}_t + \varepsilon}} \odot \boldsymbol{m}_t,$$

where the normalization $\sqrt{\mathbf{G} + \varepsilon}$ contains the matrix $\boldsymbol{G}$ that has along its diagonal the sum of squares of the past gradients w.r.t. all parameters θ_i. We vectorize $\mathbf{m}_t = \sum_{i=1}^{m} \nabla_\theta J(\theta_{t,i})$ so that we can write $\nabla_\theta J(\theta_t)$. The ε is usually a very small number that avoids a division by zero [38]. One weakness is the sum of the squared gradients in the denominator because the added terms are always positive and the summation keeps growing. Most implementations use a default value of $\eta = 1e^{-2}$.

RMSProp RMSProp algorithm modifies AdaGrad to perform better in a non-convex setting by changing the matrix $\mathbf{G}$ from a sum of squared gradients of all parameters in its diagonal to the average of squared gradients. As a result, the steep, monotonically decreasing learning rate is mitigated. Section 2.3.2.3 is altered to

$$\mathbf{G}_{ii} = \gamma \mathbf{G}_{ii} + (1 - \gamma)\boldsymbol{m}_{t,i}^2,$$

where $\mathbf{G}_{ii}$ is the diagonal element of the matrix $\mathbf{G}$ and the $\boldsymbol{m}_{t,i}$ is i^{th} the element of the vector at time t. A new hyper-parameter, the decay rate γ, is added. Hinton suggests $\gamma = 0.9$ and a learning rate $\eta = 1e^{-3}$.

Adam *Adam* [39] is another adaptive learning rate optimization algorithm. The name derives from "adaptive moment estimation". It extends RMSProp by using momentum and apply it re-scaled weights. *Adam* does not only store the exponentially decaying average of past squared gradients $\boldsymbol{v}_t$ as RMSProp but also keeps an exponentially decaying average of past gradient $\boldsymbol{\mu}_t$.

$$\begin{aligned}\boldsymbol{\mu}_t &= \beta_1 \boldsymbol{\mu} + (1 - \beta_1)\mathbf{m}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v} + (1 - \beta_2)\mathbf{m}_t^2,\end{aligned}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{v}$ are the zero-initialized estimations of the 1st (mean) and 2nd (uncentered variance) moment of the gradients [38]. The authors of *Adam* observed that during the initial time they are biased close to zero, especially when β_1 and β_2 are close to 1. Therefore, *Adam* includes bias corrections for both moments:

$$\hat{\boldsymbol{\mu}} = \frac{\boldsymbol{\mu}}{1 - \beta_1^t}, \qquad \hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{1 - \beta_2^t},$$

where t is the current training epoch. Then, the *Adam* update rule is applied:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon}\hat{\boldsymbol{\mu}}_t.$$

The authors recommend default values $\varepsilon = 1e^{-8}$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$. There are already *Adam* variants, called AdaMax and Nadam (Nesterov-accelerated Adaptive Moment Estimation).

2.3.3 Forward and Backward Propagation

In this section, we explain the forward and back propagation of a fully-connected and convolutional neural network. The forward propagation process is that the input values, along with the bias value, are multiplied by their weights and parsed to the output unit. It is taken the sum of these values and applies an activation function. The back propagation algorithm begins by comparing the actual output value by the forward propagation process to the expected value. This is the error that moves backward through the network and slightly adjusts each weights that reduces the size of the error. Both forward and back propagation are re-run thousands of times on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

2.3.3.1 Fully-Connected Neural Network

The fully-connected layer has as parameters weights and bias and a neuron is connected with every neuron in the next layer. The idea is that the fully-connected neuron forwards the weighted sum from the input connections

$$\boldsymbol{a}_j^l = f\Big(\sum_k \mathbf{W}_{j,k}^l \boldsymbol{a}_k^{l-1} + \boldsymbol{b}_j^l\Big) = f(\boldsymbol{z}_j^l), \tag{2.9}$$

where the activation $\boldsymbol{a}_j^l$ of the j^{th} neuron in the l^{th} layer depends on the k^{th} neuron in the previous layer $(l-1)^{th}$ and the input weight $\mathbf{W}_{j,k}^l$. The bias term $\boldsymbol{b}_j^l$ is usually one scalar that for the layer l^{th} layer. We want to rewrite Equation (2.9) from vector form to matrix form. We define a weight matrix $\mathbf{W}^l$ with the j^{th} and k^{th} column. Analogous to the weights we define a bias vector $\boldsymbol{b}^l$ and finally, we need to have the activation vector $\boldsymbol{a}^l$ defined. With these notations, the Equation (2.9) can be rewritten more compact:

$$\boldsymbol{a}^l = f\big(\mathbf{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\big) = f(\boldsymbol{z}^l). \tag{2.10}$$

The inner term $\boldsymbol{z}^l$ can be written according to Figure 2.4a as

$$\boldsymbol{z}^l = \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} & \mathbf{W}_{13} & \boldsymbol{b}_1 \\ \mathbf{W}_{21} & \mathbf{W}_{22} & \mathbf{W}_{23} & \boldsymbol{b}_2 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \boldsymbol{a}_3 \\ 1 \end{bmatrix}.$$

Then, the back-propagation [40] is applied, by changing weights and bias. For that, we define a small error quantity $\boldsymbol{\delta}^l$ that can occur or each neuron in each layer [25]. In other words, it adds a little change $\Delta \boldsymbol{z}_j^l$ to the neuron's weight input so that the neuron outputs $\sigma(\boldsymbol{z}_j^l + \Delta \boldsymbol{z}_j^l)$. This change propagates back through each layer in the networks:

$$\boldsymbol{\delta}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l},$$

where $\mathcal{L}$ is the loss function that is explained in Section 2.3.1. The error given at each layer is given by the gradients of the weights and bias:

$$\nabla_{\mathbf{W}}^l \mathcal{L} = \boldsymbol{a}^{l-1} \boldsymbol{\delta}^l, \tag{2.11}$$

$$\nabla_{\boldsymbol{b}}^l \mathcal{L} = \boldsymbol{\delta}^l. \tag{2.12}$$

Assume we have a training set $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), ..., (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$ of N training examples, so that we can train over network by using *Gradient Descent*. At each iteration the *Gradient Descent* updates the parameters $\mathbf{W}$, $\boldsymbol{b}$:

$$\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \nabla_{\mathbf{W}^l} \mathcal{L},$$
$$\boldsymbol{b}^l \leftarrow \boldsymbol{b}^l - \eta \nabla_{\boldsymbol{b}^l} \mathcal{L},$$

where η is the learning rate.

2.3.3.2 Convolutional Neural Network

We have explained fully-connected layers (see Figure 2.4a), where each input neuron is connected with each output neuron. This thesis focuses on Convolutional Neural Network (*CNN*), a specialization of neural networks. In *CNN* a convolution is calculated where a kernel with the depth of the input layer is sliding over input. In literature, this is often called *sliding dot product*. Different methods to calculate convolutions can be seen in Section 2.2.3.2 (the special case: dilated convolutions Section 2.2.3.3) and Section 2.2.3.4.

The forward pass through a layer each output activation is formed by

$$\begin{aligned} \boldsymbol{a}^l_{jk} &= f\Big(\sum_{j',k'} \mathbf{W}^l_{j',k'} * \boldsymbol{a}^{l-1}_{j-j',k-k'} + \boldsymbol{b}^l_{j,k}\Big) \\ \boldsymbol{a}^l &= f\big((\mathbf{W}^l * \boldsymbol{a}^{l-1}) + \boldsymbol{b}^l\big) \\ &= f(\boldsymbol{z}^l). \end{aligned}$$

As in the fully-connected case, there is given an error $\boldsymbol{\delta}(\boldsymbol{z}^l)$, that is propagated backwards [40] according to

$$\boldsymbol{\delta}^{l-1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l-1}}.$$

If we consider the section view as in Figure 2.4b the arrows show the forward propagation from $l-1$ to layer l. The calculation order is as follows, first the dashed, then the dotted and finally the full line. When the error is back-propagated from layer l to layer $l-1$ that requires the reverse order (full - dotted - dashed arrows) in Figure 2.4b. We want to show that the derivation of 2D convolution is the 2D correlation. We use the notation with indices instead of vector/matrix-notation to compare with definitions of the convolution in Equation (2.4) and of the correlation in Equation (2.5):

$$\boldsymbol{\delta}^{l-1}_{j,k} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l_{j,k}} = \sum_{j',k'} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l_{j',k'}} \frac{\partial \boldsymbol{z}^l_{j',k'}}{\partial \boldsymbol{z}^{l-1}_{j,k}} = \sum_{j',k'} \frac{\partial \boldsymbol{z}^l_{j',k'}}{\partial \boldsymbol{z}^{l-1}_{j,k}} \delta^l_{j',k'} \tag{2.13}$$

with

$$\boldsymbol{z}^l_{j'k'} = \sum_{j'',k''} \mathbf{W}^l_{j'',k''} \boldsymbol{a}^{l-1}_{j'-j'',k'-k''} + \boldsymbol{b}^l_{j',k'}$$

and the partial derivative becomes

$$\begin{aligned} \frac{\partial \boldsymbol{z}^l_{j',k'}}{\partial \boldsymbol{z}^{l-1}_{j,k}} &= \frac{\partial}{\partial \boldsymbol{z}^{l-1}_{j,k}} \sum_{j'',k''} \mathbf{W}^l_{j'',k''} \boldsymbol{a}^{l-1}_{j'-j'',k'-k''} + \boldsymbol{b}^l_{j',k'} \\ &= \boldsymbol{a}^{l-1}_{j'-j'',k'-k''} = f(\boldsymbol{z}^{l-1}_{j'-j,k'-k}), \end{aligned}$$

since only the derivations where terms at the indexes $j = j''$ and $k = k''$ in $\mathbf{W}^l_{j'',k''}$ are non-zero. The non-zero derivations imply $j' - j'' \to j' - j$ and $k' - k'' \to k' - k$. We substitute back into Equation (2.13):

$$\boldsymbol{\delta}^{l-1}_{j,k} = \sum_{j',k'} \mathbf{W}^{l}_{j',k'} \boldsymbol{\delta}^{l}_{j',k'} f(\boldsymbol{z}^{l-1}_{j,k}) \tag{2.14}$$

$$= \boldsymbol{\delta}^{l}_{j,k} * \mathbf{W}^{l}_{-j,-k} f'(\boldsymbol{z}^{l-1}_{j,k}) = \boldsymbol{\delta}^{l}_{j,k} * r_{180}(\mathbf{W}^{l}_{j,k}) f'(\boldsymbol{z}^{l-1}_{j,k}). \tag{2.15}$$

In Equation (2.14) the correlation is applied because of the change of traversing through the kernel. We use the notation to calculate a convolution on the 180 degree rotated kernel (compare Equation (2.6)), which is the correlation: $\mathbf{W}^{l}_{-j,-k} = r_{180}(\mathbf{W}^{l}_{j,k})$. Given an possible error $\delta^{l}_{i,k}$ at each neuron within a convolution layer, the weights and bias gradients are

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l}_{j,k}} = \sum_{j',k'} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l}_{j',k'}} \frac{\partial \boldsymbol{z}^{l}_{j',k'}}{\partial \mathbf{W}^{l}_{j,k}} = \sum_{j',k'} \boldsymbol{\delta}^{l}_{j',k'} \frac{\partial \boldsymbol{z}^{l}_{j',k'}}{\partial \mathbf{W}^{l}_{j,k}} \tag{2.16}$$

$$\frac{\partial \boldsymbol{z}^{l}_{j',k'}}{\partial \mathbf{W}^{l}_{j,k}} = \frac{\partial}{\partial \mathbf{W}^{l}_{j,k}} \sum_{j'',k''} \mathbf{W}^{l}_{j'',k''} \boldsymbol{a}^{l-1}_{j'-j'',k'-k''} + \boldsymbol{b}^{l}_{j',k'} \tag{2.17}$$

$$= f(\boldsymbol{z}^{l-1}_{j'-j,k'-k}). \tag{2.18}$$

$$\tag{2.19}$$

Now, we substitute back into Equation (2.16):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l}_{j,k}} = \sum_{j',k'} f(\boldsymbol{z}^{l-1}_{j'-j,k'-k})$$

$$= \boldsymbol{\delta}^{l}_{j,k} * f(\boldsymbol{z}^{l-1}_{-j,-k}) = \boldsymbol{\delta}^{l}_{j,k} * f(r_{180}(\boldsymbol{b}^{l-1}_{j,k})).$$

Finally, the derivation w.r.t. $\boldsymbol{b}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{l}_{j,k}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{l}} = \sum_{j',k'} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l}_{j',k'}} \frac{\partial \boldsymbol{z}^{l}_{j',k'}}{\partial \boldsymbol{b}^{l}}$$

since

$$\frac{\partial \boldsymbol{z}^{l}_{j'k'}}{\partial \boldsymbol{b}^{l}} = 1.$$

2.3.4 Vanishing Gradient Problem

The vanishing gradient problem [41, 42] occurs in the back-propagation process, where the error in the loss function, is back-propagated by each neuron. Certain activation functions like sigmoid squishes the large input space into a small input space. A large change in the input results in small change in the output. Let

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1^l} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2^l} \\ \vdots \end{bmatrix}, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l (\boldsymbol{a}^{l-1})^T = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,1}^l} & \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,1}^l} & \cdots \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{2,1}^l} & \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{2,2}^l} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

be the gradient of the l^{th}-layer as from our example in Section 2.2.3.2. The gradient approaches zero the more we move back through the layers, e.g. if in our network $\left|\left|\boldsymbol{\delta}^2\right|\right|_2 \ll \left|\left|\boldsymbol{\delta}^4\right|\right|_2 \ll \left|\left|\boldsymbol{\delta}^6\right|\right|_2$, then we have the vanishing gradient problem. The learning speed gets lower and lower as more we move backward until to the last layer L. LeCun et al. [43] suggest avoiding the sigmoid function in layers because the error surface is very flat at the origin and can be fast saturated. *Saturation* means that the gradient can be blocked in the back-propagation process, that means the error is not transferred to the next layer. On the basis of the proof in [44], we can show how the sigmoid activation function has a small gradient. For that, we estimate the upper bound of $\left|\left|\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}\right|\right|_2$ and $\left|\left|\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l}\right|\right|_F$ with vector and matrix norms[3]. The vector norm is denoted as $||\boldsymbol{x}||_2 = \sqrt{\boldsymbol{x}_1^2 + ... + \boldsymbol{x}_n^2}$ and the matrix norm is denoted as the Frobenius $||\mathbf{A}||_F = \sum_{i,j} |\mathbf{A}_{i,j}|^2$. An induced matrix norm is denoted as $||\mathbf{A}\boldsymbol{x}||_{a,b} = \max_x ||\mathbf{A}\boldsymbol{x}||_a$ s.t. $||\boldsymbol{x}||_b \leq 1$, where $||\cdot||_a$ is the vector norm $\in \mathbb{R}^m$ and $||\cdot||_b$ is the vector norm $\in \mathbb{R}^n$. We can apply the norms on the gradient of the bias of each layer and we denote $\boldsymbol{\Sigma}'$ as a diagonal matrix, whose diagonal is the element-wise derivation $f'(\boldsymbol{z}^l)$:

$$\left|\left|\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}\right|\right|_2 = \left|\left| \boldsymbol{\Sigma}'(\boldsymbol{z}^l)(\mathbf{W}^{l+1})^T \dots \boldsymbol{\Sigma}'(\boldsymbol{z}^{L-1})(\mathbf{W}^L)^T \boldsymbol{\Sigma}'(\boldsymbol{z}^L)(\boldsymbol{a}^L - \boldsymbol{y})\right|\right|_2 \tag{2.20}$$

$$\leq \left|\left|\boldsymbol{\Sigma}'(\boldsymbol{z}^l)\right|\right|_F \left|\left|(\mathbf{W}^{l+1})^T\right|\right|_F \dots \left|\left|\boldsymbol{\Sigma}'(\boldsymbol{z}^{L-1})\right|\right|_F \left|\left|(\mathbf{W}^L)^T\right|\right|_F \left|\left|\boldsymbol{\Sigma}'(\boldsymbol{z}^L)\right|\right|_F \left|\left|(\boldsymbol{a}^L - \boldsymbol{y})\right|\right|_2 \tag{2.21}$$

$$\leq \prod_{r=l}^{L} \left|\left|\boldsymbol{\Sigma}'(\boldsymbol{z}^r)\right|\right|_F \cdot \prod_{r=l+1}^{L} \left|\left|(\mathbf{W}^r)^T\right|\right|_F \cdot \left|\left|\boldsymbol{a}^L - \boldsymbol{y}\right|\right|_2 \tag{2.22}$$

and also on the gradient of the weights:

[3] The property of induced matrix norm: $||\mathbf{A}\mathbf{B}\boldsymbol{x}||_2 \leq ||\mathbf{A}||_F ||\mathbf{B}||_F ||\boldsymbol{x}||_2$ and $||\mathbf{A}\mathbf{B}||_F \leq ||\boldsymbol{A}||_F ||\boldsymbol{B}||_F$ hold for any matrices $\mathbf{A}$, $\mathbf{B}$ and vector $\boldsymbol{x}$ s.t. $\mathbf{A}\mathbf{B}\boldsymbol{x}$ is defined.

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} \right\|_F \leq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} \right\|_2 \left\| (\boldsymbol{a}^{l-1})^T \right\|_2 \leftrightarrow \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} \right\|_F \leq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} \right\|_2 \left\| \boldsymbol{a}^{l-1} \right\|_2 .$$

For any squared matrix as in Equation (2.22), $\|\mathbf{W}\|_F = \|\mathbf{W}^T\|_F$, thus

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} \right\|_2 \leq \prod_{r=l}^{L} \| \boldsymbol{\Sigma}'(\boldsymbol{z}^r) \|_F \cdot \prod_{r=l+1}^{L} \|\mathbf{W}^r\|_F \cdot \left\| \boldsymbol{a}^L - \boldsymbol{y} \right\|_2 .$$

Let $\zeta = sup\{\boldsymbol{\Sigma}'(\upsilon) : \upsilon \in \mathbb{R}\}$. The norm of a diagonal matrix is the largest absolute value of the elements in the matrix. The induced norm of a symmetric matrix is equal to the spectral radius. So $\| \boldsymbol{\Sigma}'(\boldsymbol{z}) \|_F \leq \zeta$ for any $\boldsymbol{z}$:

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} \right\|_2 \leq \zeta^{L-l+1} \cdot \prod_{r=l+1}^{L} \|\mathbf{W}^r\|_F \cdot \left\| \boldsymbol{a}^L - \boldsymbol{y} \right\|_2 . \tag{2.23}$$

In Figure 2.3a you can see the derivation of the sigmoid function and we can read for the sigma activation function $\zeta = 0.25$ and for tanh $\zeta = 1.0$. Hence, from Equation (2.22) and Equation (2.23) we can assume that $\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} \right\|_2$ and $\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} \right\|_F$ are very small when the network has many layers and the gradient would be small as well.

2.3.5 Parameter Initialization

Network parameter initialization does not only have an impact on the convergence speed but the wrong initialization can prone the network to the vanishing/exploding gradient problem. Weights $\mathbf{W}_0$ are usually initialized close to zero.

Truncated Normal Initialization This initialization is similar to a normal distribution except these values which are more than 2 standard deviations away from them mean are discarded. It is a standard deviation $\frac{1}{\sqrt{n_{in}}}$, where $\mathbf{W}_0 \sim n_{in}$ is the number of inputs to the given layer. For example, the sigmoid function is saturated (compare Figure 2.3a) and your neuron will not learn. So, the truncated normal distribution does not have this issue at least for the initialization [43].

Xavier Initialization It [45] is named after Xavier Glorot and sometimes also called Glorot. It was found after analyzing the back-propagated gradients

$$\mathbf{W}_0 \sim \sqrt{\frac{2}{n_{in} + n_{out}}} \mathcal{N}(0, 1),$$

where n_{out} describes the number of output units.

He Initialization [46] This is the preferred initialization compared to the Xavier initialization [45] when ReLU activation (see Figure 2.3b) functions are used throughout the neural networks. He recommends initializing the weights by

$$\mathbf{W}_0 \sim \sqrt{\frac{2}{n_{in}}}\mathcal{N}(0,1).$$

Biases are usually initialized with zeros.

2.3.6 Overfitting

Overfitting refers to the problem that the training data shows up excellent results, while the validation data becomes more worse in every iteration. The underlying reason is that the network size is too big and less complexity of the model is required.

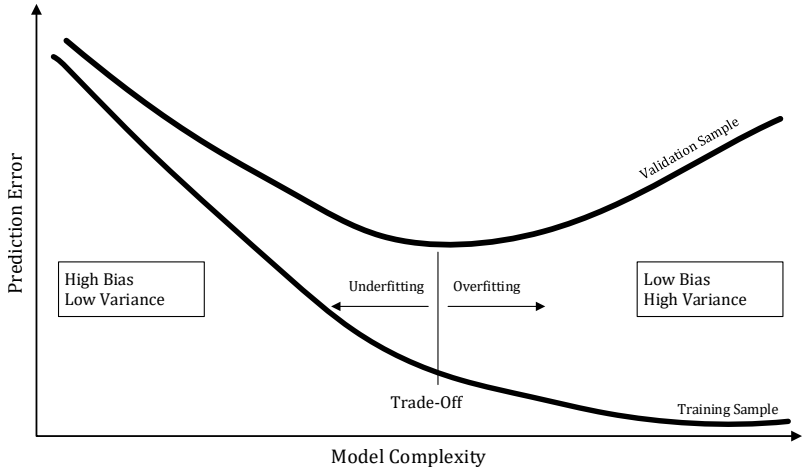


Figure 2.11: Bias-variance-tradeoff of a model.

2.3.6.1 Bias-Variance-Tradeoff

This tradeoff [47] tells us how well the model is fitting the training data and how well it can generalize to new unseen data. It describes the problem of keeping bias and variance in balance (Figure 2.11):

- High bias (underfitting) can cause that a neural network does not fit the relation between features and label well.
- High variance (overfitting) can cause that a neural network does fit the features too tight to the label so that the neural network does not generalize well. One possible remedy is data augmentation, that is more detailed described in Section 4.2.3.

2.3.6.2 Early Stopping

When the error on the training set tends to be reduced by every epoch, the error on the validation decreases usually as well. It can happen that the validation error rises at some point, once the network starts to overfit the training set. To avoid overfitting, the training can be stopped as soon as the validation error rises [48].

2.3.6.3 Weight Decay

Weight decay is another method to reduce overfitting when there is a fix amount of training and a fixed network. There are two weight decay regularization methods [25] $L1$ [49] and $L2$[4] [50]. The idea is to add an extra term to the cost function denoted as regularization term:

$$J(\boldsymbol{\theta}) = \mathcal{L}_0 + \underbrace{\frac{\lambda}{N}\sum_{i,j}|\mathbf{W}^l_{i,j}|}_{\text{L1 regularization term}}, \qquad J(\boldsymbol{\theta}) = \mathcal{L}_0 + \underbrace{\frac{\lambda}{2N}\sum_{i,j}\left|\left|\mathbf{W}^l_{i,j}\right|\right|^2_F}_{\text{L2 regularization term}}$$

where $\mathcal{L}_0$ can be any unregularized cost function and the $L1/L2$-norm of all weights in the network is the regularization term that is scaled by a $\lambda > 0$ divided by the size of the training set is N. If λ is small, the original cost function $\mathcal{L}_0$ is minimized, but when λ is large, small weights are preferred to learn. This regularization can compromise minimizing the original cost function and finding small weights.

Both regularization terms have several limitations and are investigated by Zou and Hastie [51], but one does not dominate the other one. While the $L1$-norm has the sparsity property that might shrink less important feature coefficient to zero, but it cannot group similar features, the $L2$-norm does only minimize these coefficients instead of setting them to zero and has always a unique solution.

2.4 Computing Stereo Matching Costs with Convolutional Neural Networks

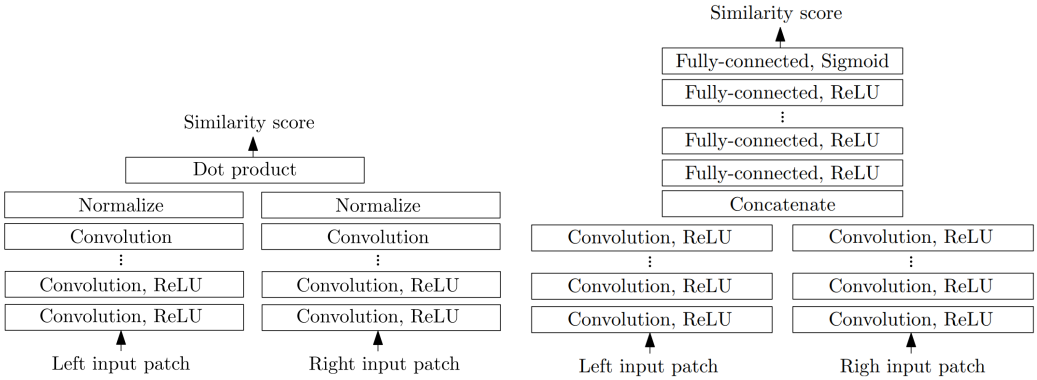
Zbontar and LeCun [6] introduced a machine learning approach to extract depth information from a rectified image pair. Therefore, image patches of size of 5×5 are compared from each image respectively with a *CNN*. The more similar these patches of the image pair are, the lower the costs. Occluded pixels would yield very high costs.

In this section two methods are compared along the epipolar line to each other, which calculate the stereo matching cost. One is faster, while the other is more accurate. The output of both is not ready for the final result, so that further steps are necessary. For

[4] Frobenius Norm: $||\mathbf{A}||^2_F = \sum_{i,j}|\mathbf{A}_{i,j}|^2$

learning the *CNN*, two cost functions are used, the Cross-Entropy (*CE*) and the Hinge Loss (*HL*).

The output of both networks are not satisfying, so they refined the output of the *CNN* mainly with two post-processing methods: cross-based cost aggregation and Semi Global Matching (*SGM*). The network is trained and tested on the datasets KITTI [52] and Middlebury [3].



(a) Fast Architecture. **(b)** Accurate Architecture.

Figure 2.12: Overview of both architectures [6].

2.4.1 Unary Network

Siamese network [53, 54] is an architecture consists of 2 identical neural networks, the unary *CNN*, each taking the left and right input image of a stereo pair. The parameters are usually shared by each layer of both networks. This means that the weights are exactly the same. The shared weights makes it possible to learn the similarity. After feature extraction, both networks are joined at their outputs. The purpose of this architecture is to learn the similarity of both inputs what is useful to find pixel correspondences as for disparities. If no correspondences are found that can be interpreted as occlusions. Both unary *CNN*s have several convolutional layers with ReLU as non-linearity.

There are two architectures "fast" and "accurate", which distinguish of joining the output of both unary *CNN*s. The fast architecture in Figure 2.12a compares the normalized output vectors of both unary networks using the Euclidean dot product as similarity measure. For the accurate architecture (Figure 2.12b) the dot product layer is replaced by fully connected convolutional layers. These layers have as well ReLU as non-linearities and the last layer has the non-linearity sigmoid to produce the output probabilities prediction. The left input patch $\mathbf{P}^L$ is fix and for every disparity d the right input patch $\mathbf{P}^R$ is shifted:

$$\boldsymbol{C}^{CNN}_{i,j,d} = f(\mathbf{P}^L_{i,j}, \mathbf{P}^R_{i,j-d}),$$

so that the matching costs $\boldsymbol{C}^{CNN}$ can be formed.

2.4.2 Loss Functions

Zbontar and LeCun claimed that the binary cross-entropy results better for accurate architecture while it is not applicable for the fast architecture, so they took the hinge loss. The basic concept of both loss function are explained in Section 2.3.1. Although, they would prefer using the same loss for better comparison the final results. For the fast architecture they use as loss the Hinge Loss (HL) and for the accurate architecture they use the Cross-Entropy (CE):

Loss of the fast architecture: To minimize the error a HL is considered. Two image patches are chosen, where one belongs to class $\hat{y}_+$ and the other to the class $\hat{y}_-$. The hinge loss can be defined as $\max(0, m + \hat{y}_- \hat{y}_+)$. If the output $\hat{y}_+$ is greater than $\hat{y}_-$ by a margin $m = 0.2$, then the loss is zero.

Loss of the accurate architecture: Let $\hat{y}$ be the output of one training example and y the class related to $\hat{y}$. The label is $y = 1$, if the training example belongs to the class. Otherwise, the label will be 0. Now, the binary CE loss can be applied $y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$.

2.4.3 Post-processing

The output of fast and accurate CNN methods needs to be improved. Therefore, for finding corresponding pixels cross-based aggregation [55] is used that uses flexible window size to find similar pixels with the same disparity value. Then, the matching costs of the cross-based-aggregation are refined with Semi Global Matching (SGM) [56] that enables the smoothness constraint to ensure that there are no jumps in the disparity map on homogeneous areas.

2.4.3.1 Cross-based Cost Aggregation

Cross-based cost aggregation [7] is another method that takes information about neighboring pixels. Instead taking the average cost over a fixed window that assume a constant depth withing this window, cross-based cost aggregation selects the neighboring pixels for every pixel. The method begins to construct an upright cross at each pixel position. The left arm p_l extends as long as the two conditions must hold for selecting the correct neighbors:

1. The intensities at pixel $p = i, j \in dom\ \mathbf{I}^L$ and at $p_l = k, l \in dom\ \mathbf{I}^R$ should be similar: $|\mathbf{I}^L_{i,j} - \mathbf{I}^R_{k,l}| < \tau$.
2. Only pixels within a certain distance are compared.

The right, bottom and top arms are constructed in the same way. If the four arms are known, we can define the support region $\mathbf{U}(p)$ of all positions lying on p's vertical arm. When the arms in horizontal and vertical direction are found the support region $\mathbf{U}(p)$ is computed to close the area between the arms.

Zhang et al. [7] suggest that the aggregation on both images from a stereo pair should be considered. Let $\mathbf{U}^L$ and $\mathbf{U}^R$ be the support regions (see Figure 2.13) in the left and right image. The matching cost is averaged between the left and right support region $\mathbf{U}^L \cup \mathbf{U}^R$. For the accurate architecture in Figure 2.12b) this Let $\mathbf{U}$ be the combined support region:

$$\begin{aligned}\mathbf{U}_{i,j} &= \{q | q \in \mathbf{U}^L_{i,j}, q - d \in \mathbf{U}^R_{i,j-d}\} \\ \boldsymbol{C}^0_{i,j,d} &= \boldsymbol{C}^{CNN}_{i,j,d} \\ \boldsymbol{C}^t_{i,j,d} &= \frac{1}{|\mathbf{U}_{i,j}|} \sum_{k,l \in \mathrm{U}_{i,j}} \boldsymbol{C}^{t-1}_{k,l,d},\end{aligned}$$

where t denotes the current iteration. Since the support regions can overlap the results can change at every iteration. The iterations are skipped for the fast architecture, see Figure 2.12a, because of the expensiveness to compute it.

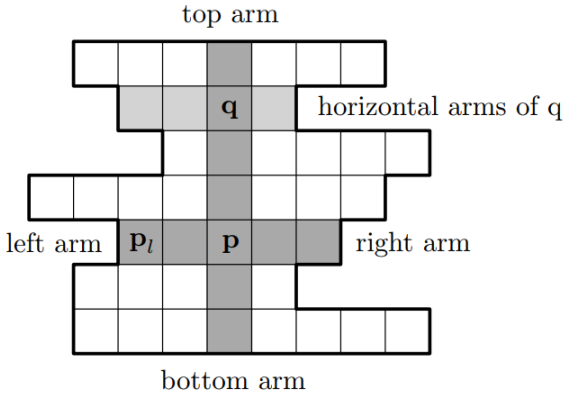


Figure 2.13: A support region for a position p at $i, j \in dom\ \mathbf{I}^L$ is the union of all vertical and horizontal arms [7].

2.4.3.2 Semi Global Matching

The disparity map from Cross-Based Aggregation in Section 2.4.3.1 $\boldsymbol{C}$ can be refined by using Semi Global Matching (SGM) [56]. An energy function $E(\mathbf{D})$ needs to be defined that relies on the disparities $\mathbf{D}$:

$$E(\mathbf{D}) = \sum_{i,j} \left(\boldsymbol{C}^4_{i,j,\mathbf{D}_{i,j}} + \sum_{l,k \in \mathcal{N}_{i,j}} \lambda_1 \cdot \mathbf{1}\{|\mathbf{D}_{i,j} - \mathbf{D}_{l,k}| = 1\} + \sum_{l,k \in \mathcal{N}_{i,j}} \lambda_2 \cdot \mathbf{1}\{|\mathbf{D}_{i,j} - \mathbf{D}_{l,k}| > 1\} \right),$$

where $\mathbf{1}(\cdot)$ is the indicator function. $E(\mathbf{D})$ can be split into 3 terms. The first term of the function sums all pixel-wise matching costs over the whole image. The second term

penalizes neighboring pixels that differs by 1. The third term adds a larger penalty for all pixels with neighbors that have a different disparity. In that way, discontinuities are allowed if pixel-wise matching is stronger than the penalty, e.g. if the texture signalize a discontinuity. The third term indirectly connects all pixels within the image and hence this makes the function global. In *SGM* the energy is minimized in several directions r:

$$\begin{aligned}\boldsymbol{C}^r_{i,j,d} = \boldsymbol{C}^4_{i,j,d} - \min_k \boldsymbol{C}^r_{i-r_i,j-r_j,k} + \min\big\{&\boldsymbol{C}^r_{i-r_i,j-r_j,d}, \boldsymbol{C}^r_{i-r_i,j-r_j,d-1} + p_1,\\ &\boldsymbol{C}^r_{i-r_i,j-r_j,d+1} + p_1, \min_k \boldsymbol{C}^r_{i-r_i,j-r_j,k} + p_2\big\}.\end{aligned}$$

The first pixel are defined as $C^4_{i,j,d}$. The second term is subtracted to avoid too large costs [57]. Then, the minimum is taken: The costs of the pixels in the direction r. The costs with one higher and lower disparity is computed. The last values is the minimum cost from the the pixels in direction r over all disparities. The penalty parameters p_1 is constant for the small change in the neighborhood and p_2 is adaptive for the large changes in the neighborhood are added. A lower penalty permits an adaption to slanted surfaces. Hirschmüller [56] suggested 16 directions. Zbontar and LeCun [6] claimed only to minimize into 4 directions, 2 horizontal and 2 vertical directions. Adding more directions does not improve the accuracy. An average over all directions is calculated to obtain the final result. The matching cost $\boldsymbol{C}^r_{i,j,d}$ in direction r is achieved through:

$$\boldsymbol{C}^{\text{SGM}}_{i,j,d} = \frac{1}{4}\sum_r \boldsymbol{C}^r_{i,j,d}.$$

The disparity d that minimizes $\boldsymbol{C}^{SGM}_{i,j,d}$ forms the disparity map $\mathbf{D}_{i,j}$:

$$\mathbf{D}_{i,j} = \arg\min_d \boldsymbol{C}^{SGM}_{i,j,d}.$$

Since *SGM* cannot handle occlusions [58] the disparity map is post-processed with Left-Right-Cross-Checking (*LRC*) explained in Section 2.6.1. The occlusion map for the left or for right stereo image can be calculated from the disparity maps in the dataset as in Equation (2.24).

2.5 End-to-End Training of Hybrid CNN-CRF Models for Stereo

Hybrid CNN-CRF [8] learns the similarity of stereo images first as we have investigated in Section 2.4.1 and then apply a Conditional Random Fields (*CRF*) for refining the disparity map. This approach takes advantage of both *CNN* and *CRF*. In Figure 2.14, the model is split into different blocks and is described in the following sections in detail. Basically, it takes two input images and computes the similarity to construct a stereo correlation

volume. The stereo correlation is fed into the *CRF*. The *CRF* takes contrast sensitive weights for the depth discontinuity and pairwise interactions in a local neighborhood are for smoothness. The inference is approximated with a fixed number of iterations of a linear programming relaxation based approach. The stereo correlation implementation use the prior knowledge for occlusion detection.

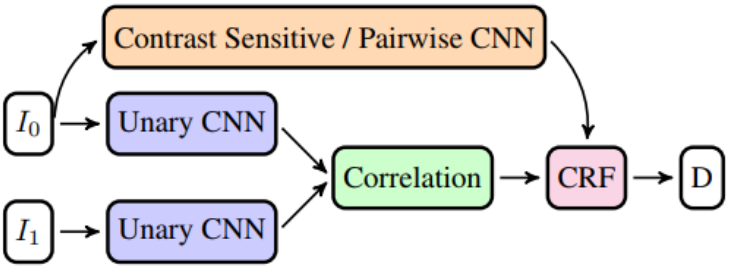


Figure 2.14: A *Unary-CNN* is the *CNN* for each image. In the *correlation* layer the features of are compared. The matching cost becomes a unary cost volume becomes the unary cost of the *CRF* [8].

2.5.1 End-to-End Learning

End-to-end is a terminology in learning, where all parameters are trained jointly as compared to step-by-step. The classical way was to give some input then extract features and perhaps there some more intermediate steps necessary. In end-to-end all intermediate steps are involved and the output is produced.

The advantage is that the hyper-parameters do not have to be determined for each intermediate step as we have seen in Section 2.4, where Zbontar and LeCun [59] used cross-based aggregation and Semi Global Matching (*SGM*). On the one hand, end-to-end learning architectures save to set hyper-parameters of several post-processing methods. On the other hand, not all methods cannot be integrated in the back-propagation algorithm.

2.5.2 Unary Netnwork and Shared Weights

Unary network with 3 or 7 layers with 100 kernels for each layers are used. The kernel size of the first layers differs by 3×3. For later layers 2×2 sized kernels are chosen. As activation function, the tanh (Section 2.2.1) is selected, because of the ease of training: it keeps the output bounded and there is no (batch-)normalization layer required [60, 61].

The *shared weights* are used during training between both unary *CNN*s. This ensures that network learns the similarity between both input images.

It is assumed that left $\mathbf{I}^L$ and the right $\mathbf{I}^R$ are rectified, where the epipolar lines correspond to the image rows. The left image is the reference image and we search for corresponding pixels in the right image. The disparity of a pixel $i, j \in dom\ \mathbf{I}^L$ is represented

by a discrete label $d \in \mathcal{Y} = \{0, ..., Y-1\}$. Then, dense image features are extracted, denoted as $\boldsymbol{\phi}^L = \boldsymbol{\phi}(\mathbf{I}^L; \boldsymbol{\theta}_1)$ and $\boldsymbol{\phi}^R = \boldsymbol{\phi}(\mathbf{I}^R; \boldsymbol{\theta}_1)$ with the shared parameters $\boldsymbol{\theta}_1$.

2.5.3 Stereo Correlation

To form a cost volume, for each pixel these extracted features are correlated at all possible disparities. The confidence $\boldsymbol{P}_{i,j,d}$, where $\boldsymbol{P}_{i,j,d} : \Omega \times \mathcal{Y} \rightarrow [0, 1]$ means that how well a window around a pixel i, j in the first image matches the window around pixel $i, j + d$. The *cross-correlation* offers to combine the extracted features $\boldsymbol{\phi}^L_{i,j}$, $\boldsymbol{\phi}^R_{i,j}$ of both unary-*CNN*s

$$\boldsymbol{P}_{i,j,d} = \frac{e^{\langle \boldsymbol{\phi}^L_{i,j}, \boldsymbol{\phi}^R_{i,j+d} \rangle}}{\sum_{d \in \mathcal{Y}} e^{\langle \boldsymbol{\phi}^L_{i,j}, \boldsymbol{\phi}^R_{i,j+d} \rangle}} \qquad \forall i, j \in \Omega, \ \forall d \in \mathcal{Y}$$

that transfers the features to probability, because it outputs the softmax (see Section 2.2.1) of the corresponding feature vectors. This improves to train the joint network, because scales the unary-costs. The best matching disparity is determine according to the winner-takes-all strategy, where

$$\boldsymbol{P}_{i,j} \in \arg\max_d \boldsymbol{C}_{i,j,d}$$

is selected for the full model.

2.6 Occlusion Detection

In this section, we discuss methods that detects occlusions exclusively. Compared to stereo matching, there is not a huge number of methods, but occlusions are very often detected on the fly. We want to pay attention on the methods, which only focus on detecting occlusions.

In computer vision, occlusions are areas hidden by other objects. We focus on stereo images [62], where two images are taken from the same scene. Both cameras have the same distance to the object but have a distance to each other. There are pixels in one image that cannot be seen in the other image, so these pixels are occluded.

In Figure 2.15, the dotted lines are the lines of sight from both camera centers (C, C'). The line of sight from C is interrupted by an object and hence cannot capture the same position on the scene as the line of sight from C'. This position is occluded by an object. In the stereo setup, occlusions occur at the border of objects.

2.6.1 Left-Right-Consistency-Check

Left-Right-Cross-Checking (LRC) [6, 63, 64] is one of the most accurate methods to compute occlusions from pre-computed disparities. According to Egnal [14] LRC has the highest hit and lowest false positive rate. A pixel i, j is occluded if the constraint is

fulfilled

$$|\mathbf{D}^L_{i,j} - \mathbf{D}^R_{i,j-\mathbf{D}^L_{i,j}}| > \upsilon, \qquad \upsilon \in \mathbb{R}, \tag{2.24}$$

where $\mathbf{D}^L$ and $\mathbf{D}^R$ are the disparities maps from the left and right image respectively. Disparity maps are show in Figure 1.3 from left-to-right and right-to-left, where for left-to-right the left image is chosen as reference frame and vice versa. For each pixel i, j of the left-to-right disparity map the matching disparity on the right-to-left disparity map is looked up. If the disparities in both disparity maps are bigger than a threshold υ [65], this should be an occluded pixel.

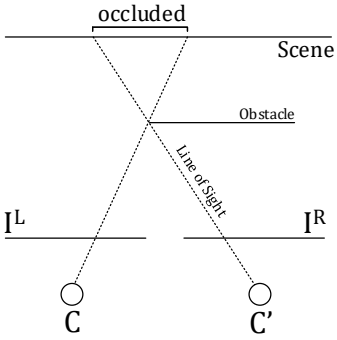
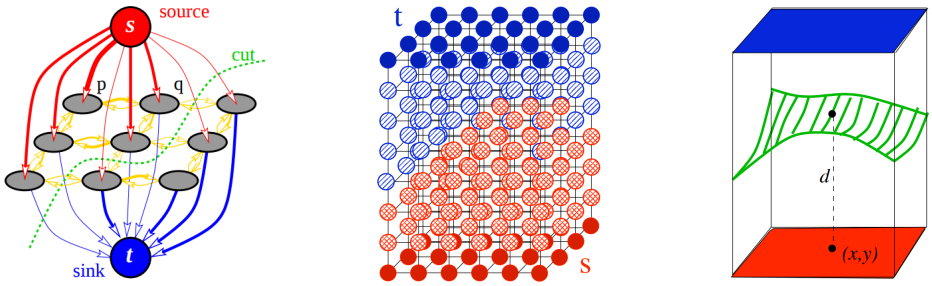


Figure 2.15: A scene is captured by two cameras. The occluded area is enclosed by the two lines of sight.

2.6.2 Graph Cut Algorithm for Point Correspondences and Occlusions

Kolmogorov and Zabih [13] developed a graph cut algorithm, that adapt the correspondence problem into a maximum flow/minimum cut problem. This is another method to handle occlusions in stereo images. Let $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ a weighted graph with its vertices $\mathcal{V}$ and edges $\mathcal{E}$, while two terminal vertices named source and sink $\{s, t\}$. In Figure 2.16a is a simple graph, where each pixel is a gray node and connected to each other. The weights from the pixel nodes to the source or sink are infinity and are not considered in the minimum cut. It is required to find a minimum cut $\mathcal{C} = \mathcal{V}^s, \mathcal{V}^t$, that partitions the vertices either to the sink $s \in \mathcal{V}^s$ or to the source $t \in \mathcal{V}^t$. We distinguish between the pixels of the left and right stereo image, L and R respectively. The costs of a cut are determined of the weights of the edges between a vertex in $\mathcal{V}^s$ and $\mathcal{V}^t$. The minimum cut problem can be efficiently solved by using the maximum flow theorem [66], which only needs low-order polynomial time as worst-case. Computing the minimum cut problem is equivalent to solving the maximum flow. The theorem of Ford and Fulkerson [66] states that the maximum flow in $\mathcal{G}$ is equal to the costs of the minimum cut. In other words, a minimum cut has the set of edges whose maximum capacities were reached in the maximum flow solution.



(a) Simple Graph. **(b)** Graph (d=0..k). **(c)** Disparities.

Figure 2.16: Different Graph $\mathcal{G}$ with each pixel has one node per disparity [9].

To find corresponding pixels, the disparity have to be calculated and the graph can be imagined as in Figure 2.16b, where on the vertical axis the different disparities of the corresponding pixels are stored. Dark blue and red are the source/terminal nodes. The lighter. The result of the graph cut is green as in Figure 2.16c what yields the final disparity map. They assume that all disparities occur in a range of $[0, k]$:

$$\mathcal{A} = \{< p, q > | p = (p_x, p_y) \in \mathcal{L}, q = (q_x, q_y) \in \mathcal{R}, p = q_y, 0 \leq q_x - p_x \leq D,\ D \in \mathbb{N}_0\}. \tag{2.25}$$

Instead of assigning disparities to each pixel, a configuration needs to be defined, where a function $\xi : \mathcal{A} \to \{0, 1\}$ maps the assignment $\mathcal{A}$ to 1 if the assignment is active and 0 otherwise. The set of all currently active assignments in the configuration of ξ as $\mathcal{A}(\xi) = \{a \in \mathcal{A} | \xi_a = 1\}$ and $N_p(\xi)$ the active assignments of the pixel p in the given configuration. To be sure that a pixel from the left image has only one corresponding pixel in the right image the constraint of *uniqueness* must hold

$$|N_p(\xi)| \leq 1, \qquad \forall p \in \mathcal{L} \cup \mathcal{R},$$

whereas the criterion for an occluded pixel is if $|N_p(\xi)| = 0$. Further, a set of all assignments $\mathcal{A}^\alpha$ is defined that contains pairs of resembled pixels having the disparity α. To this, the disparity $d : \mathcal{A} \to \mathbb{R}$ as $d(a) = d(< p, q >) = q_x - p_x$ is defined. With these notions, a configuration ξ' is within one α-expansion of ξ, if $\mathcal{A}(\xi') \subset \mathcal{A}(\xi) \cup \mathcal{A}^\alpha$. This means that some α-assignment might become inactive and any other α-assignment might become activated. It is not possible to activate assignments with a different α. It might happen that a deactivated α-assignment that no other assignment incorporate with this pixel p, e.g. if $|N_p(\xi')| = 0$, it can marked occluded. As a next step, we have to define the energy for a configuration. For non-unique configurations the energy is infinity and for unique configuration the energy is

$$E(\xi) = E_{occ}(\xi) + E_{data}(\xi) + E_{smooth}(\xi),$$

where energy must be extended with $E_{occ}(\xi)$ penalty term for making a pixel occluded. The occlusion term is necessary to limit the occluded pixels:

$$E_{occ}(\xi) = \sum_{p \in L \cup \mathcal{R}} C_p \cdot \mathbf{1}(|N_p(\xi)| = 0) \tag{2.26}$$

where $C_p > 0$ is the penalty, the higher C_p the more penalty and $\mathbf{1}(\cdot)$ is 1 if the argument is true otherwise false. The sum over all pixels shows the symmetry of this method, because the occluded pixel in the left as well in the right image is penalized.

The data term compares the intensities between corresponding pixels. For an assignment a is the pixel intensity compared as $D(a) = (\mathbf{I}^L_{p_y,p_x} - \mathbf{I}^R_{q_y,q_x})^2$, where $\mathbf{I}^L_{p_y,p_x}$ is the intensity value at pixel p of the left image. The data term is formulated over the sum of all assignments by the difference of the pixel intensities:

$$E_{data}(\xi) = \sum_{a \in A(\xi)} D(a). \tag{2.27}$$

The smoothness term involves the local neighborhood and is the most complex part of the energy function. The neighborhood $\mathcal{N}$ considers only pairs s.t. a_1 and a_2 have the same disparity and i.e. p and p' are neighbors for the assignment pairs $\{\langle p, q\rangle, \langle p', q'\rangle\}$ and $d(\langle p, q\rangle) = d(\langle p', q'\rangle)$. Thus, the smoothness term can be expressed as

$$E_{smooth}(\xi) = \sum_{\{a_1,a_2\} \in \mathcal{N}} V(a_1, a_2) \cdot \mathbf{1}(\xi(a_1) \neq \xi(a_2)) \tag{2.28}$$

and states if the disparities of two comparing pixels are equal, then the smoothness penalty is zero, otherwise the energy keeps its positive value. The algorithm selects one α after another finds a unique configuration withing an α-expansion move. If the energy is decreased, then we go there. Unless there is no α which decreases energy, the algorithm terminates.

2.6.3 SymmNet - First End-to-End Model to learn Occlusions

SymmNet [10] is the first publication to learn occlusions by using exclusively a *CNN*. There are many approaches to find information about occlusions via disparity maps of the left and right image. They learned occlusions by using an hour-glass architecture with a high number of parameters (approx. 5 Mio.). As can be seen in Figure 2.17 the hourglass architectures consists of an contracting and afterwards an expanding part, where the images are downsampled and upsampled to the original size to keep the pixelwise prediction. They decided to take data-sets with binocular images because monocular images bear to much uncertainty in regard to scene geometry, camera settings and pictorial structure.

SymmNet predicts 2 occlusion maps, one for the left image and one for the right image. Ang Li and Zeijan Yuan claim that they outperform other methods described in their paperf (MonoNet, SiameseNet, AlterNet, HalfNet, LRCNet) on the datasets Middlebury, Sintel and Sceneflow. Due to SymmNet is the first machine learning approach to learn occlusions, they had to improvise several other approaches:

MonoNet is the more simple SymmNet that predict only one occlusion map either left-to-right or right-to-left. It gives only a clue of the object edges, where usually are the most occluded areas, but it is also a rise to fake occlusion.

SiameseNet has one unary CNN for each input image and uses shared weights. This architecture is often used for stereo matching. The shared weights prevent to learn distinctive information what would be necessary for occlusion detection.

AlterNet is the idea to compute the occlusion for one view, while train the network periodically swapping the input images. They complained about bad predictions on the right view.

HalfNet describes an approach with 2 indepent networks, which only gets binocular images as input, but the feature channel length of SymmNet. The results are almost as good as the SymmNet hour-glass architecture for both views, but numerically it is slightly worse.

LRCNet is an approach where the prediction layer in SymmNet is replaced by a regression layer to compute a disparity map and then apply *LRC* on these disparities. This is the only network which calculates the disparities and the performance is rather poor. Occluded areas are not recognized and have many holes of non-occluded pixels.

2.6.3.1 Architecture

The architecture relies on the model FlowNet [67, 68], that is an end-to-end architecture predicting optical flow. As input the stereo images are stacked (see Table 2.1), so 2 RGB images have an input depth of 6. Then, for the contracting part, the images are downsampled about the half size, since the striding parameter is set to 2. They repeat the downsampling procedure 6 times in order to progressively increase the receptive field and sub-sample the spatial size of the feature maps. During downsampling the feature size is doubled every time from 16 until to a maximum of 512 and during upsampling the feature size is decreased incrementally. After each downsample layer follows a 2D convolution layer to smoother the results. To keep the fine local information, skip connections from the downsampling to the upsampling are introduced. The concatenations in the skip connections is replaced by an addition to keep the feature size as in the corresponding downsampling layer. The last upsampled output is concatenated with the input images. For the prediction are 4 kernels, whereas 2 for each view is normalized with softmax.

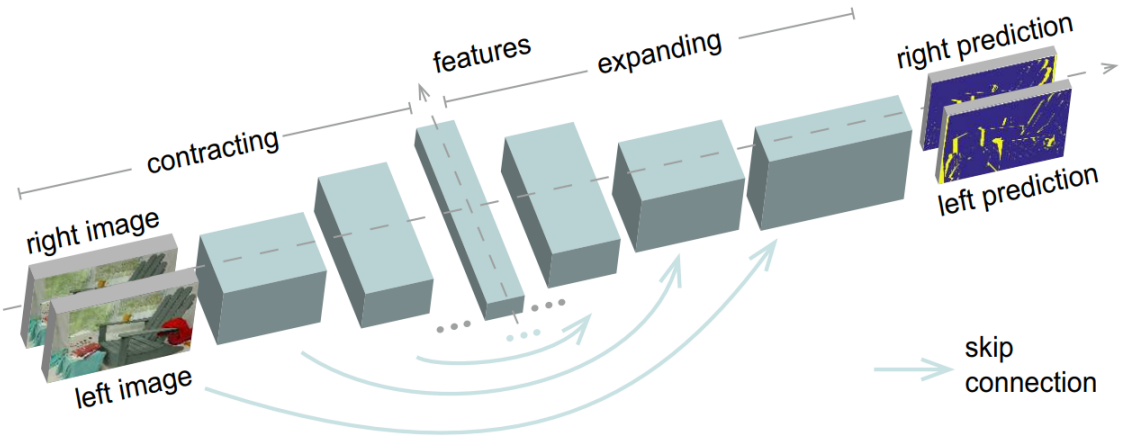


Figure 2.17: Architecture overview of SymmNet [10].

Name	Kernel	Str.	Ch I/O	OutRes	Input
Input					
input			6/6	H x W	image pair
Contracting					
dwnsp1	8x8	2	6/16	1/2H x 1/2W	input
conv1	3x3	1	16/16	1/2H x 1/2W	dwnsp1
dwnsp2	6x6	2	16/32	1/4H x 1/4W	conv1
conv2	3x3	1	32/32	1/4H x 1/4W	dwnsp2
dwnsp3	6x6	2	32/64	1/8H x 1/8W	conv2
conv3	3x3	1	64/64	1/8H x 1/8W	dwnsp3
dwnsp4	4x4	2	64/128	1/16H x 1/16W	conv3
conv4	3x3	1	128/128	1/16H x 1/16W	dwnsp4
dwnsp5	4x4	2	128/256	1/32H x 1/32W	conv4
conv5	3x3	1	256/256	1/32H x 1/32W	dwnsp5
dwnsp6	4x4	2	256/512	1/64H x 1/64W	conv5
conv6	3x3	1	512/512	1/64H x 1/64W	dwnsp6

Name	Kernel	Str.	Ch I/O	OutRes	Input
Expanding					
upsp5	4x4	2	512/256	1/32H x1/32W	conv6
iconv5	3x3	1	256/256	1/32H x1/32W	upsp5+conv5
upsp4	4x4	2	256/128	1/16H x1/16W	iconv5
iconv4	3x3	1	128/64	1/16H x1/16W	upsp4+conv4
upsp3	4x4	2	64/64	1/8H x1/8W	iconv4
iconv3	3x3	1	64/32	1/8H x1/8W	upsp3+conv3
upsp2	4x4	2	32/32	1/4H x1/4W	iconv3
iconv2	3x3	1	32/16	1/2H x1/2W	upsp2+conv2
upsp1	4x4	2	16/16	1/2H x1/2W	iconv2
iconv1	3x3	1	16/14	H x W	upsp1+conv1
upsp0	4x4	2	16/8	H x W	iconv1
iconv0	3x3	1	14/8	H x W	upsp0⊕input
Prediction					
pr	3x3	1	8/4	HxW	iconv0

Table 2.1: Detailed SymmNet architecture: The plus sign + is the addition operation, ⊕ is the concatenation operation in skip connection.

2.6.3.2 Loss Function and Training

They jointly train the total binary cross-entropy loss for the left and right view:

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = -\frac{1}{2}\Big(w_o^L \sum_{i,j} \mathbf{1}(\mathbf{Y}_{i,j}^L = 1) \log(\hat{\mathbf{Y}}_{i,j}^L)) + w_{\bar{o}}^L \sum_{i,j} \mathbf{1}(\mathbf{Y}_{i,j}^L = 0) \log(1 - \hat{\mathbf{Y}}_{i,j}^L) + w_o^R \sum_{i,j} \mathbf{1}(\mathbf{Y}_{ij}^R = 1) \log(\hat{\mathbf{Y}}_{i,j}^R) + w_{\bar{o}}^R \sum_{i,j} \mathbf{1}(\mathbf{Y}_{i,j}^R = 0) \log(1 - \hat{\mathbf{Y}}_{i,j}^R) \Big),$$

where $\mathbf{Y} = \mathbf{Y}^L \cup \mathbf{Y}^R$ is the ground-truth of the from left-to-right and right-to-left, $\mathbf{1}(\cdot)$ is the indicator function,

$$w_c = \frac{1}{\log(q_c)} \tag{2.29}$$

is a weight to make the loss adapt to the unbalanced number of o and $\bar{o}$ pixels with q_c is the proportion of class $c \in \{o,\ \bar{o}\}$ and an $\epsilon = 1.2$ is added to the $\log(\epsilon + q_c)$. They trained their neural network with a synthetic SceneFlow [69]. It also delivers disparity

maps, so that the occlusions can be computed with the *LRC* Section 2.6.1 to use random batch, in Section 4.2.3, as data augmentation to prevent over-fitting. They also fine-tuned their training with the Middlebury dataset [3, 70], that is mainly used in this work, see Section 4.1.1. They split it into training and validation set to conduct a 10-fold cross-validation. They could achieve a f-score of 0.828, while 1.0 is the best score. Without pre-training the network they only could a achieve an f-score of 0.66. The Kolmogorov and Zabhit graph cut approach [71] only scores 0.6. The f-score is computed from the Precision-Recall (*PR*) metric.

2.7 Summary

In this chapter, the related work according to occlusions and it's state-of-the-art methods have been presented. We started from standard algorithms, e.g. the census transform and Sum of Squared Differences (*SSD*). We have established the relation of *SSD* and correlation that is used in Convolutional Neural Network (*CNN*). If we rotate the kernel by 180 degrees, we can transform a 2D convolution to a 2D correlation. Then, we have presented basic elements of the *CNN*, such as the neuron itself. We also described different kinds of convolutional layers and in particular the receptive field. Furthermore, we have explained the activation functions and their derivations and in Section 2.3.4 we know that the sigmoid function is sub-optimal to use in intermediate layers. We have pointed out different loss functions and how to train a neural network.

We have treated several approaches to learn point correspondences or disparity and the connection to occlusions. There are two types to solve this problem: One is to train a neural network to find correspondences and put a lot of effort on post processing techniques. Many hyper parameters needs to be set. The other is in end-to-end manner, where the whole approach is trained jointly.

Finally, we have summed up three methods that handles occlusion. Apart from Left-Right-Cross-Checking (*LRC*), that is also used to compute the ground-truth from disparity maps. Disparity maps are often contained in known datasets and are usually very precise. It is also very often used to post-process the final results. There is a graph cut approach by Kolmogorov and Zabih that uses point correspondences to calculate the occluded pixels and an end-to-end *CNN* method called SymmNet. SymmNet outperforms the graph cut approach and is therefore the best approach to compute occlusions.

3 Methods

Contents

In this chapter, we describe our four methods in detail. In Figure 3.1, we give a brief overview, where we separate the direct method from the methods based on the stereo correlation. For the stereo correlation, we utilized the implementation of the *CNN-CRF* [8], that calculates a cost volume of how similar pixels are of a image stereo pair. In Section 2.5 we have described the background to calculate the stereo correlation.

First, we start with explaining a method that we call direct method, which uses dilated convolutions in the unary *CNN*s. The naming convention points to learn occlusions directly from the input images instead of from the stereo correlation. The direct method can be better compared to the "Cor+2D" method of the stereo correlation because both methods use 2D dilated convolutions.

Second, we treated another stereo correlation method "Cor+3D" based on the 3D convolutions (Section 2.2.3.4). Instead of 2D dilated convolutions we use one 3D convolution layer as the aggregation. It also uses the Cross-Entropy (CE) as loss function.

Third, all those models predict the probability of each pixel whether it is occluded or non-occluded. In order to classify these pixels, a global threshold must be found so that every probability that is bigger than this threshold is assigned to occluded (o) or non-occluded ($\bar{o}$).

At the end of this chapter you will find a summary of our methods. The differences of each method will be highlighted and compared to each other.

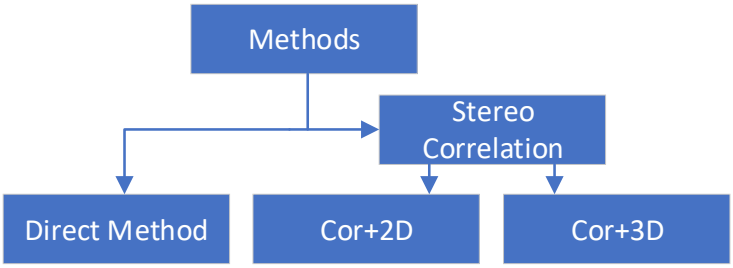


Figure 3.1: Overview of our methods.

3.1 Direct Method - learn Occlusions without Stereo Correlation

The direct method is our first approach to define the problem of occlusion detection as a learning problem. We consider this method as competitor of SymmNet (see Section 2.6.3), that we have discussed thoroughly in the Section 2.6.3. They put on an hour-glass architecture. We decided to use a unary network architecture (see Section 2.4.1 as well in Figure 3.2) with dilated convolutions. The use of dilated convolution avoids to downsample and upsample again the images because no striding is applied. This saves some layers, whereas SymmNet needs to add 2D convolution layer after downsampling to obtain smoother results. The output of both unary *CNN* is concatenated and gives us the derivation of both input images. After concatenation several layers are added to achieve a final occlusion map.

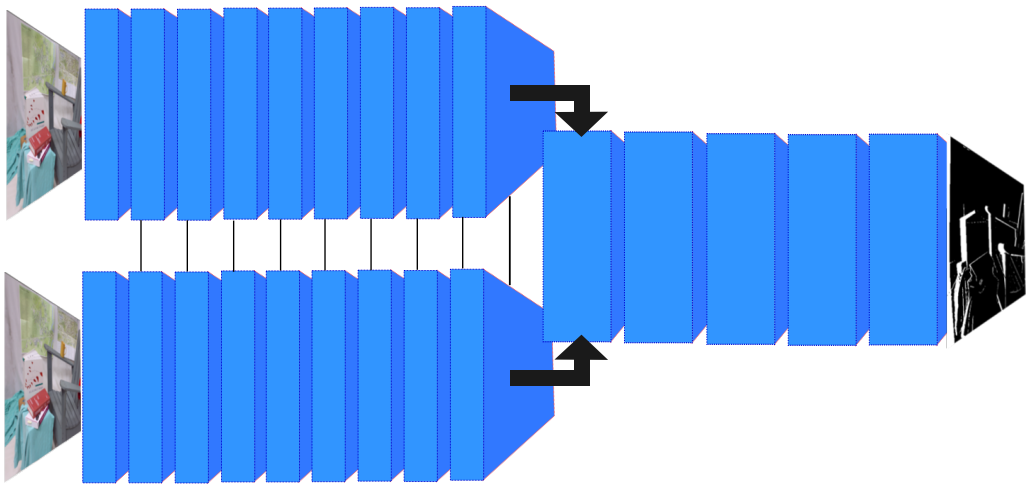


Figure 3.2: Direct method network architecture with 2 stereo input images and predicts an occlusion map.

Unary CNN In Table 3.1 is a detailed view of the network architecture that starts with the two input images (left, right) with the size of 304×380. The Siamese network architecture demands to share the weights and bias for each layer to learn the similarity of both images. Beside that, 2D convolutions (see Section 2.2.3.2) are taken with a kernel size of 3×3 with the activation function ReLU (see Section 2.2.1). Every third layer the dilation rate (see Section 2.2.3.3) is increased by one, whereas the dilation rate (see Section 2.2.3.3) is specified to be the power of 2, i.e. $2^{r-1}, r \in \mathbb{N}$. Fisher et al. [72] showed that they can outperform 2D convolutional networks with dilated convolutions without increasing the model's depth. The key of dilated convolution is to keep the spatial information. In Section 2.2.3.3, we have described the basic dilated convolution. In order to group the dilated convolutions by their dilation rate r we have to modify Equation (2.7) to:

$$(\mathbf{U}^g *_r \mathbf{H}^g)_{i,j} = \sum_{m,n} \mathbf{U}^g_{(i+r-1)-m,(j+r-1)-n} \cdot \mathbf{H}^g_{m,n}, \tag{3.1}$$

where g denotes the i^{th} layer in the group g. Within a group g the dilation rate r is the same. We decided to have a maximum rate $r = 4$ since the image has only 304×380 pixels. The receptive field on the last dilated convolutional layer of each unary *CNN* has a size of 15 pixels.

Aggregation The output of the unary *CNN*s is concatenated. Concatenating gives better results than subtracting the right image from the left images. At this point we have one concatenated output and the occlusions must be learned. In Table 3.1 we call this procedure aggregation, where 5 layers with 2D convolutions with 3×3 kernels and 64 output kernels are applied.

On the last layer *pred*, we decided to use sigmoid (see Section 2.2.1) as activation function, which squashes the predictions of each pixel between 0 and 1 and can be interpreted as probabilities. This is important for the *CE* loss function which compares probability distributions.

Binary Cross-Entropy The binary cross-entropy is a loss, that measures the performance of our model. A perfect model would have a loss of 0. For example, if the prediction $\tilde{\mathbf{Y}}_{i,j}$ has a probability of 0.02 and the actual observation label $\mathbf{Y}_{i,j} = 1$, would give a high loss value. The binary *CE* loss basically consists of two classes

$$CE(\mathbf{Y}, \tilde{\mathbf{Y}}) = -\frac{1}{n} \sum_{i,j}^{n} \left[\mathbf{1}\{\mathbf{Y}_{i,j} = 1\} \log(\tilde{\mathbf{Y}}_{i,j}) + \mathbf{1}\{\mathbf{Y}_{i,j} = 0\} \log(1 - \mathbf{Y}_{i,j}) \right],$$

where only the loss for the corresponding class is calculated. If the label $\mathbf{Y}_{i,j} = 1$ (occluded) only the first term of the summation is active and the other term is inactive and does not count to the entropy. If $\mathbf{Y}_{i,j} = 0$ the first term is zero and the second term is active for entropy calculation. The loss is normalized by n which is the total

number of pixels of an input image. More details and deviation of the *CE* can be found in Section 2.3.1.

Name	Kernel	Activation	Dil.	Ch I/O	OutRes	Input
Input						
input				3/3	1/6H × 1/6W	image pair
Siamese						
dilate1_1	3x3	ReLU	1	3/32		input
dilate1_2	3x3	ReLU	1	32/32		dilate1_1
dilate1_3	3x3	ReLU	1	32/32		dilate1_2
dilate2_1	3x3	ReLU	2	32/32		dilate1_3
dilate2_2	3x3	ReLU	2	32/32		dilate2_1
dilate2_3	3x3	ReLU	2	32/32		dilate2_2
dilate4_1	3x3	ReLU	4	32/32		dilate2_3
dilate4_2	3x3	ReLU	4	32/32		dilate4_1
dilate4_3	3x3	ReLU	4	32/32		dilate4_2
Concatenation						
concat				2×32/64		dilate4_3l ⊗ dilate4_3r
Aggregation						
aggr_1	5x5	ReLU	1	64/64		concat
aggr_2	5x5	ReLU	1	64/64		aggr_1
aggr_3	3x3	ReLU	1	64/64		aggr_2
aggr_4	3x3	ReLU	1	64/64		aggr_3
aggr_5	3x3	ReLU	1	64/64		aggr_4
Prediction						
pred	3x3	sigmoid	1	64/1	1/6H × 1/6W	aggr_5

Table 3.1: Direct method Siamese architecture with shared weights. The $\oplus$ sign is the concatenation operation to concatenate the outcome of the left. The prediction layer *pred* has the sigmoid activation function to generate probability.

Costs Weighting In our training set there are ten times less occluded than non-occluded pixels. Thus, the loss takes ten times more often non-occluded into account. We have to balance the data so that the loss takes both classes equally into account. We calculate the ratio by taking the number of occluded/non-occluded divided by total number of pixels. The ratio of occluded q_o and non-occluded $q_{\bar{o}}$ is calculated as:

$$q_o = \frac{\sum_{i,j} |\mathbf{Y}_{i,j} - 1|}{\sum_i |\mathbf{Y}_{i,j} - 1| + \sum_{i,j} \mathbf{Y}_{i,j}}, \qquad q_{\bar{o}} = \frac{\sum_{i,j} \mathbf{Y}_{i,j}}{\sum_{i,j} |\mathbf{Y}_{i,j} - 1| + \sum_{i,j} \mathbf{Y}_{i,j}},$$

where the number of labels are 1 for occluded pixels or are 0 for non-occluded. We multiply the ratio of the other class (occluded (o) or non-occluded ($\bar{o}$)) to the loss (inverse

frequency). In the SymmNet loss [10] a logarithmic class weight w_c is introduced:

$$w_c = \frac{1}{\log(q_c)},$$

where classes $c \in \{o, \bar{o}\}$ and an ϵ is added to $\log(\epsilon + q_c)$. The binary *CE* has to be modified towards

$$CE(\mathbf{Y}, \tilde{\mathbf{Y}}) = \frac{1}{n} \sum_{i,j}^{n} \left[w_o \big(\mathbf{1}\{\mathbf{Y}_{i,j} = 1\} \log(\tilde{\mathbf{Y}}_{i,j}) \big) + w_{\bar{o}} \big(\mathbf{1}\{\mathbf{Y}_{i,j} = 0\} \log(1 - \tilde{\mathbf{Y}}_{i,j}) \big) \right],$$

where the logarithm yields negative values for input values in range of $0 < q_c < 1$ under the assumption that there are always occluded pixels. Thus, the minus is removed in front of the sum (compare Section 3.1).

Valid Mask A valid mask masks out invalid pixels in the loss. Invalid pixels have the disparity value infinity in the depth map, what means that no valid disparity could be found for this pixel. If we train the network with the images we take the valid pixels from the ground-truth provided in the Middlebury dataset. More information about this dataset can be found in Section 4.1.1. If we train the network with random batches, then we have to compute the labels for this patch from the left-to-right and right-to-left depth-maps and the invalid pixels are determined only from the left-to-right depth-map. So, the valid mask $\mathbf{M}_{i,j}$ is defined as

$$\mathbf{M}_{i,j} = \begin{cases} 1 & \text{if } \mathbf{Y}_{i,j} \text{ is valid,} \\ 0 & \text{else.} \end{cases} \tag{3.2}$$

The *CE* must be modified to multiply $\mathbf{M}_{i,j}$ with the inner term of the *CE*:

$$CE(\mathbf{Y}, \tilde{\mathbf{Y}}) = \frac{1}{n} \sum_{i,j}^{n} \mathbf{M}_{i,j} \left[w_o \big(\mathbf{1}\{\mathbf{Y}_{i,j} = 1\} \log(\tilde{\mathbf{Y}}_{i,j}) \big) + w_{\bar{o}} \big(\mathbf{1}\{\mathbf{Y}_{i,j} = 0\} \log(1 - \tilde{\mathbf{Y}}_{i,j}) \big) \right]. \tag{3.3}$$

3.2 Pretrained Stereo Correlation

Stereo correlation method differs from the direct method from using less layers and dilations for the aggregation. The advantage of this stereo correlation function is that disparities are taken into account. A detailed description can be found in the related work Section 2.5. We extracted the forward and backward CUDA implementation[1] from Knöbelreiter et al. [8]. Therefore, we need to implement a tensorflow operation that is

[1] github.com/VLOGroup/cnn-crf-stereo

callable in Python source code and can be used for learning. It follows the *aggregation*, that can be 2D convolutional layers, 3D convolutional layers and entropy.

Name	Kernel	Activation	Str.	Ch I/O	OutRes	Input
Input						
input				3/3	1/6H×1/6W	image pair
Simaese Network						
2dconv1	3×3	tanh	1	100/100		input
2dconv2	2×2	tanh	1	100/100		2dconv1
2dconv3	2×2	tanh	1	100/100		2dconv2
2dconv4	2×2	tanh	1	100/100		2dconv3
2dconv5	2×2	tanh	1	100/100		2dconv4
2dconv6	2×2	tanh	1	100/100		2dconv5
2dconv7	2×2	tanh	1	100/100		2dconv6
Stereo Correlation						
corr	3×3	softmax	1	100/disp	1/6H×1/6W	2dconv7

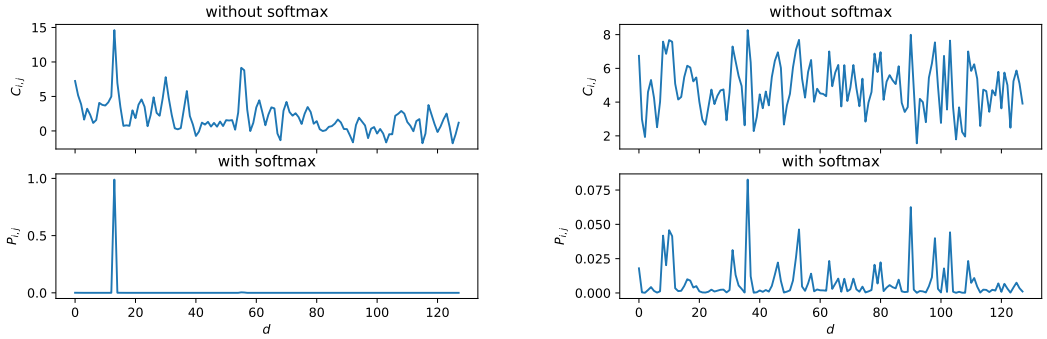
Table 3.2: Network configuration of the *CNN* with stereo correlation [8].

Stereo Correlation (For-/Backward) The *stereo correlation* layer computes a dot product with the output of the two unary *CNN*s. The left image is the fixed image and the right image is adapted to the disparities. Disparity 0 does not shift the right image, so a dot product is calculated. Disparity $d = -1$ shifts the right tensor to the right by one. The most right column vanishes. If the disparity is $d = -2$, then the right tensor is shifted by 2 to the right and the last 2 columns vanishes. After each shift, we have computed a matrix with pixel-wise dot products. This matrix is appended over the depth of the resulting matrix from the previous shift, that forms a cost volume. The output of left and right unary *CNN*s the features $\{\boldsymbol{\phi}^L_{i,j,d}, \boldsymbol{\phi}^R_{i,j,d} \in \mathbb{R} | i, j \in \boldsymbol{\phi}^L \; dom \; \Omega, \; d \in \mathcal{D}, \mathcal{D} = \{0, ..., 128\}\}$ are the extracted features of the left and right image. The dot product is mathematically expressed as cost volume $\boldsymbol{C}_{i,j,d} = \langle \boldsymbol{\phi}^L_{i,j}, \boldsymbol{\phi}^R_{i,j+d} \rangle$. We define a loss $\mathcal{L}(\boldsymbol{\phi}) = \frac{1}{2} ||\boldsymbol{C}||^2$ and derive w.r.t. $\boldsymbol{\phi}$:

$$
\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}^L, \boldsymbol{\phi}^R) &= \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^L}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^R} \right) \\
\nabla_{\boldsymbol{\phi}^L} \mathcal{L}(\boldsymbol{\phi}^L, \boldsymbol{\phi}^R) &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{C}_{i,j,d}} \cdot \frac{\partial \boldsymbol{C}_{i,j,d}}{\partial \boldsymbol{\phi}^L_{i,j}} \\
&= \boldsymbol{C}_{i,j,d} \cdot \boldsymbol{\phi}^R_{i,j+d} \\
\nabla_{\boldsymbol{\phi}^R} \mathcal{L}(\boldsymbol{\phi}^L, \boldsymbol{\phi}^R) &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{C}_{i,j,d}} \cdot \frac{\partial \boldsymbol{C}_{i,j,d}}{\partial \boldsymbol{\phi}^R_{i,j+d}} \\
&= \boldsymbol{C}_{i,j,d} \cdot \boldsymbol{\phi}^L_{i,j}.
\end{aligned}
$$

For normalization, we altered Equation (3.4) by using softmax over the disparity d

$$\boldsymbol{P}_{i,j,d} = \frac{e^{\langle \boldsymbol{\phi}^L_{i,j}, \boldsymbol{\phi}^R_{i,j+d} \rangle}}{\sum_{d \in D} e^{\langle \boldsymbol{\phi}^L_{i,j}, \boldsymbol{\phi}^R_{i,j+d} \rangle}} \qquad \forall i, j \in \Omega, \forall d \in \mathcal{D}. \tag{3.4}$$



(a) 2 matching pixels of $\boldsymbol{\phi}^L_{i,j}$ and $\boldsymbol{\phi}^R_{i,j+d}$. **(b)** 2 not matching pixels of $\boldsymbol{\phi}^L_{i,j}$ and $\boldsymbol{\phi}^R_{i,j+d}$.

Figure 3.3: Correlation of two matching pixels between $\boldsymbol{\phi}^L_{i,j}$ and $\boldsymbol{\phi}^R_{i,j+d}$.

Tensorflow needs to back propagate the stereo correlation. For the standard operations, tensorflow uses automatic differentation [29, 73], that calculates the derivatives of the symbolic graph of the model. We also needed to include the CUDA/C++ stereo correlation back propagation so that tensorflow can compute the differentiation. We checked the back propagation numerically by approximated the outcome with different disparities the 1st derivation[2]. Hence, we could show an accuracy of $\epsilon = 1e^{-6}$ decimal digits, what is the float point precision. The forward differences is calculated

$$\mathcal{L}'(\phi) = \frac{\mathcal{L}(\phi_{i,j,d} + \epsilon) - \mathcal{L}(\phi_{i,j,d})}{\epsilon} + O,$$

which means we subtract the actual point from the actual point plus a small $\epsilon \in \mathbb{R}$ and that divided by the ϵ. For this reason it is an approximation, we would need to add an additional term O, so that it is no approximation anymore. We tested the implementation on images sizes from 3×4 to 100×100 with a Gaussian distribution (mean $\mu = 0$, standard deviation $\sigma = 1$, compare Figure 4.3) and compared the outcome from the CUDA implementation with the numerical approximation. The numerical approximation on the 100×100 test image with 32 disparities takes around 47.3 minutes on the CPU Intel i7 960 @ 3.20GHz. In contrast, the CUDA implementation takes only 0.37 seconds.

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.approx_fprime.html`

3.2.1 Three Methods based on the Stereo Correlation

At this point, we have a cost volume of the stereo correlation. The depth of the cost volume depends on the number of disparities. If we compute 128 disparities, the depth of the volume is 128. If the pixels with a shift of disparity are matching the correlation at this position is high in this volume. If the pixels do not match, the correlation is low.

In this section we have different post-correlation methods investigated to refine the correlation and finally obtain the binary classification. We also have improvements opposed to the direct method: For counteracting the possible vanishing gradient problem (see Section 2.3.4) we use lReLU as activation function instead of ReLU. In order to give a better initialization we used He instead of the Xavier/Glorot initialization (see Section 2.3.5).

We used the *CE* (see Equation (2.29)) from direct method for aggregations with the *CE* as loss functions (as in Section 3.2.1.1 and Section 3.2.1.2).

3.2.1.1 Dilated 2D Convolution Aggregation

The idea of this aggregation is the same as of our direct method in Section 3.1. The difference is that we apply these layers on the stereo correlation instead of directly on the RGB channels of the input images.

This time we decided again against a hourglass architecture, to avoid the downsampling and upsampling procedure (compare SymmNet in Section 2.6.3), that is not given in dilated convolutions.

Similar to the architecture of the direct method (see Table 3.1) we use dilated convolutions, but lReLU as activation function for each layer (see Table 3.3). The receptive field has also a size of 15 pixels. The reason for the similar architecture is to have a comparison from learning directly and from stereo correlation.

Name	Kernel	Act	Dil.	Ch I/O	OutRes	Input
Aggregation						
2dconv1	3×3	lReLU	1	disp/32	1/6H×1/6W	corr
2dconv2	3×3	lReLU	1	32/32		2dconv1
2dconv3	3×3	lReLU	1	32/32		2dconv2
2dconv4	3×3	lReLU	2	32/32		2dconv3
2dconv5	3×3	lReLU	2	32/32		2dconv4
2dconv6	3×3	lReLU	2	32/32		2dconv5
2dconv7	3×3	lReLU	4	32/32		2dconv6
2dconv8	3×3	lReLU	4	32/32		2dconv7
2dconv9	3×3	lReLU	4	32/32		2dconv8
Prediction						
pred	3x3	sigmoid	1	32/1	1/6H×1/6W	2dconv9

Table 3.3: Input is the soft stereo correlation from Table 3.2.

3.2.1.2 3D Convolution Aggregation

The stereo correlation produces a cost vector for every pixel. Hosni et al. [74] refined this cost volume with a guided kernel. Based on the idea to refine the cost volume, we use 3D convolutions (see Section 2.2.3.4). In Table 3.4, we have as input the cost volume with the depth of the number of disparities. As kernel, we use a $3 \times 3 \times 3$ kernel, initialized with He (see Section 2.3.5). The third dimension is the depth of the kernel and influences the number of output channels, that is usually #disparities minus (depth-1). A bigger kernel size would increase the number of learnable parameters and would be sub-optimal for a small training set.

Name	Kernel	Act	Dil.	Ch I/O	OutRes	Input
Aggregation						
3dconv1	$3 \times 3 \times 3$	lReLU	1	disp/disp-2	1/6H×1/6W	$\boldsymbol{P}$
Prediction						
pred	3×3	sigmoid	1	disp-2/1	1/6H×1/6W	3dconv1

Table 3.4: 3D convolution aggregation. The input is the soft stereo correlation from Table 3.2.

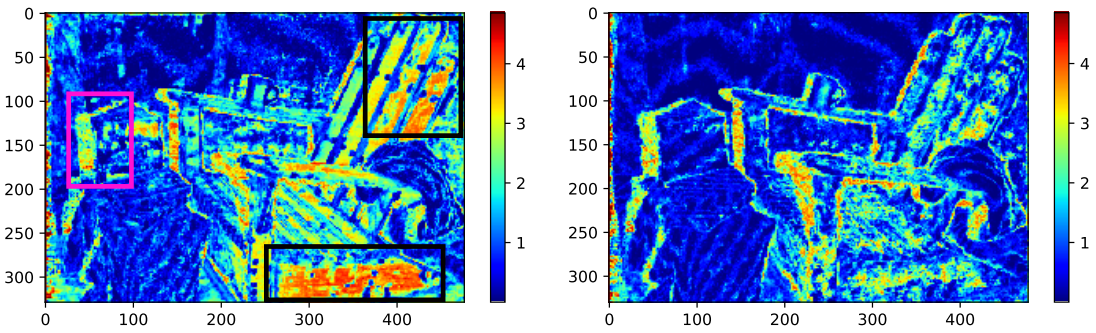
Shannon's entropy is the measure of information. If we want to measure the entropy of 2 identical patches, the entropy will be high. Otherwise, if there are differences the entropy will be low. The Shannon entropy $\mathcal{H}$ is defined as

$$\mathcal{H}(\boldsymbol{P}_{i,j}) = -\sum_{d} \boldsymbol{P}_{i,j,d} \log(\boldsymbol{P}_{i,j,d}),$$

where the uncertainty over the $\boldsymbol{P}_{i,j,d}$ confidence are calculated. For numerical reasons, a little $\varepsilon = 1e^{-6}$ is added within the logarithm $\log(\max(\boldsymbol{P}_{i,j,d}, \varepsilon))$ since the logarithm is not defined for ≤ 0.

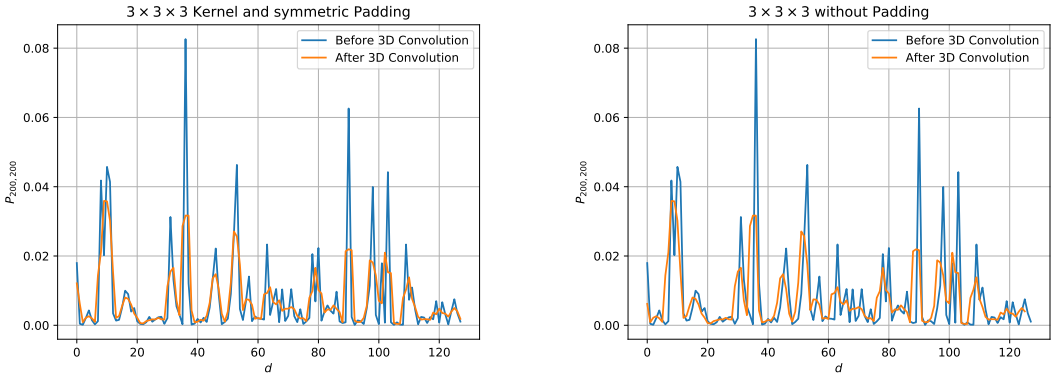
The output of the forward propagation can have negative values because the layers used the non-linearity tanh as activation function. We must transform the output to probabilities, and thus we use the softmax function over the disparities. The stereo correlation CUDA implementation offers the possibility to take 3 and 7 layers. We focus on better detection results rather than on inference speed and therefore we take into account more layers which would slow down. In Figure 3.4, we can visually compare the cost volume quality. For that, we calculated the entropy on the channels of each pixel. The higher the entropy value (at most red) it is more likely an occluded pixel. The stereo correlation with 7 layers seems to have better entropy values on the correct pixel positions in Figure 3.4b.

We have an example of a 3D convolution in Figure 3.5a with a $3 \times 3 \times 3$ kernel with ones divided by 27, what calculates the 3D convolution in a $3 \times 3 \times 3$ neighborhood. If we would not use padding as in Figure 3.5b, we would not have some noise at the beginning and end of the feature vector. The kernel values are smaller than one and multiplied with the entries from the input tensor to finally sum all multiplications up. The 3D convolution results in a smaller value per disparity than the original disparities.



(a) Entropy on stereo correlation with 3 layers. **(b)** Entropy on stereo correlation with 7 layers.

Figure 3.4: Entropy on the cost volume. On the left hand-side we marked the purple area for correct occluded area and black for incorrect occluded areas.



(a) With padding. **(b)** Without padding.

Figure 3.5: 3D convolution on $\boldsymbol{P}_{i,j,d}$ and a $3 \times 3 \times 3$ kernel with $1/27$ per entry value.

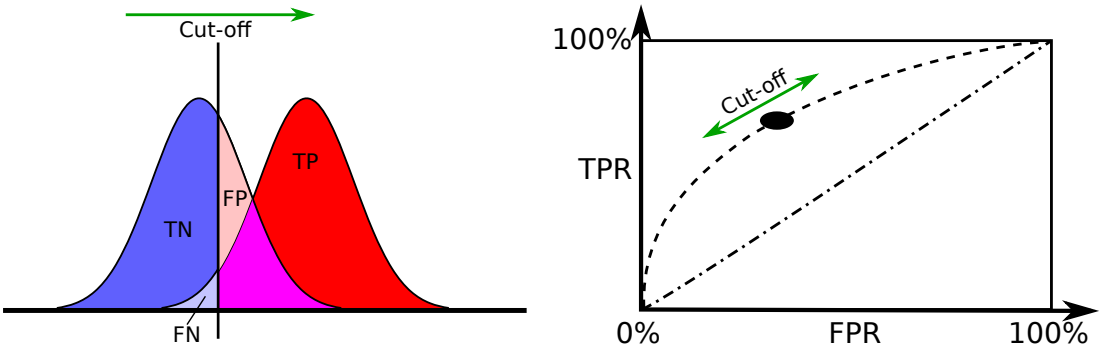
3.3 Global Thresholding

A global threshold needs to be calculated that is applied after the prediction. This threshold classify the pixels into occluded (o) and non-occluded ($\bar{o}$). One of the simplest method to do this is the Receiver Operating Characteristic (ROC) [75], that classify pixel-wise the prediction by using many cut-off points ϑ (see Figure 3.6a). Since our predictions have confidence between 0 (very unconfident to be occluded) and 1 (very confident) that a pixel is occluded. A cut-off point can have a value between 0 and 1. For example if we have a cut-off point of 0.5 then the confident value greater than 0.5 is occluded and non-occluded otherwise. For every possible cut-off point different categories are determined:

- True Positive (TP) - The pixel is classified as o and is labeled as o.
- True Negative (TN) - The pixel is classified as $\bar{o}$ and is labeled as $\bar{o}$.
- False Positive (FP) - The pixel is classified as o and is labeled as $\bar{o}$.

- False Negative (FN) - The pixel is classified as $\bar{o}$ and is labeled as o.

If the cut-off point minimizes FP and FN which are the wrongly classified pixels, the ROC curve should grow to the left upper corner in Figure 3.6b. This would indicate the best classification results because the False Positive Rate (FPR) (wrong classified) is small and the True Positive Rate (TPR) (correct classified) is large. The goal is to find a cut-off that maximize the TP and TN classified pixels, which are the correct classified pixels.



(a) Ratio of different categories. **(b)** ROC curve.

Figure 3.6: Relation of cut-off points and ROC curve [11].

3.3.1 Receiver Operating Characteristic Curve

In order to determine the ROC curve (Figure 3.6), we need to define the True Positive Rate (TPR) and the False Positive Rate (FPR). On the y-axis of the ROC plot is the TPR defined:

$$TPR = \frac{TP}{TP + FN}$$

for all positive predicted values. It describes the ratio of correctly classified pixels to be occluded divided by all pixels which are classified to be occluded. On the x-axis is the False Positive Rate (FPR) computed which is defined as

$$FPR = \frac{FP}{FP + TN},$$

where it is the ratio between the number of wrongly classified occluded pixels divided by the total number of wrongly classified pixels. For calculating the fractions, we need to transfer the probabilities values $\tilde{\mathbf{Y}}_{i,j}$ comparable to the labels $\mathbf{Y}_{i,j}$:

$$\hat{\mathbf{Y}}_{i,j} = \begin{cases} 1 & \text{if } \tilde{\mathbf{Y}}_{i,j} > \vartheta, \\ 0 & \text{else.} \end{cases}$$

From that point, predictions $\tilde{\mathbf{Y}}_{i,j}$ and labels $\mathbf{Y}_{i,j}$ can be compared with a $L1$-norm that

is shown in the Table 3.5:

$\hat{\mathbf{Y}}_{i,j}$	$\mathbf{Y}_{i,j}$	$\lvert\hat{\mathbf{Y}}_{i,j} - \mathbf{Y}_{i,j}\rvert$	$(1 - \lvert\cdot\rvert)$	$(1 - \lvert\cdot\rvert)\mathbf{Y}_{i,j}$	$(1 - \lvert\cdot\rvert)(1 - \mathbf{Y}_{i,j})$	$\lvert\cdot\rvert\mathbf{Y}_{i,j}$	$\lvert\cdot\rvert(1 - \mathbf{Y}_{i,j})$	
0	0	0	1	0	**1**	0	0	TN
0	1	1	0	0	0	**1**	0	FN
1	1	1	0	0	0	0	**1**	TP
1	0	0	1	**1**	0	0	0	FP

Table 3.5: The categories can be calculated via the $L1$-norm, where $|\cdot| = |\hat{\mathbf{Y}}_{i,j} - \mathbf{Y}_{i,j}|$. In each row, the where a **1** is written, indicates {TN,FN,TP,FP}.

In order to obtain the correct comparison result, we have to invert the result of the $L1$-norm. If the values of $\hat{\mathbf{Y}}_{i,j}$ and $\mathbf{Y}_{i,j}$ are equal:

$$1 - |\hat{\mathbf{Y}}_{i,j} - \mathbf{Y}_{i,j}| = \begin{cases} 1 & \text{if } \hat{\mathbf{Y}}_{i,j} = \mathbf{Y}_{i,j}, \\ 0 & \text{else.} \end{cases}$$

3.3.2 Optimal Decision Threshold

The optimal decision threshold can be read in the *ROC* plot. The closer the *ROC* curve is to the upper-left corner the more precise is the algorithm. Hence, we want to find the minimum distance

$$l(\vartheta) = \sqrt{(1 - TPR(\vartheta))^2 + FPR(\vartheta)^2}, \qquad \vartheta \in \{0.001k : k \in \mathbb{N}_0, 0 \leq k \leq 1e^3\},$$

by calculating the Pythagoras between x and the inverted y-axis. The the optimal threshold is at

$$\vartheta^* = \arg\min_{\vartheta} l(\vartheta).$$

Then, the classification of the pixels are given by $\tilde{\mathbf{Y}}$ at ϑ^*:

$$\hat{\mathbf{Y}}_{i,j} = \begin{cases} 1 & \text{if } \tilde{\mathbf{Y}} > \vartheta^*, \\ 0 & \text{if } \tilde{\mathbf{Y}} \leq \vartheta^*. \end{cases}$$

3.4 Summary

In this chapter, we have introduced our 4 models. We can distinguish our models between 2 categories: One is not based on the computation of disparity (direct) and the others are based on the computation of disparity. None of these methods use post-processing to improve the results and are end-to-end architectures to train all intermediate steps jointly.

The receptive field of the unary *CNN*s of the direct method and the 3D convolution with dilated convolutions for aggregation has the same size of 15 pixels. We have found out that concatenation of the both unary *CNN*s of the direct method yields shorter training

time and at least the same results as if we would subtract/add the output of the unary CNNs.

We have also checked the quality of the stereo correlation and found that the 7-layers (see Figure 3.4) are better suited to predict occlusions on the base of the uncertainty of the Shannon entropy.

The output of our models are probabilities about how confident our model is that a pixel is occluded, but this is not the final detection. We propose to calculate the Receiver Operating Characteristic (ROC) curve to determine an optimal threshold. This threshold classify pixels to be occluded or non-occluded and hence this gives the final occlusion map.

4 Evaluation

Contents

In this chapter, we present the evaluation of models. We describe our dataset and how we augment our data. The performance of our models is measured by Receiver Operating Characteristic (ROC), Area Under Curve (AUC) and the pixel-wise comparison with the ground-truth. To train our models, we used the following hardware/software components: Ubuntu 16.04.06, Nvidia GTX 980 4 GB, CUDA 8, CuDNN 6, Tensorflow [29]. To train the networks we used *Adam* [39] and used various learning rates from $\eta = 1e^{-2}$ to $\eta = 1e^{-5}$, for the first momentum $\beta_1 = 0.9$ and for the second momentum a value of $\beta_2 = 0.99$.

4.1 Datasets

We used two datasets to train our models, whereas our focus is on the Middlebury 2014 dataset. The second dataset, SceneFlow (Monkaa), is needed to overcome the over-fitting problem with the small Middlebury dataset. Both datasets have their properties (image size, number of images, ground-truth) that we briefly describe in this section.

4.1.1 The Middlebury 2014 Dataset

The Middlebury 2014 dataset contains 33 indoor scenes and was published by Scharstein and Hirschmueller [3]. It also provides as ground truth disparity maps and masks where occlusions and invalid pixels are marked. Beside that, there exist two stereo image pairs per scene with *perfect* and *imperfect* rectification. While the imperfect rectification shows higher errors over rectification, especially in regions of high-frequency texture. The y-offset

is larger and can vary over some rows, so that the corresponding pixel is not in the same row in the other stereo image. The image size differs per image pair, but the full size has around 1984×2872 pixels. The labels in Figure 4.1 are prepared, so that the grey areas are occluded and the rest are non-occluded.

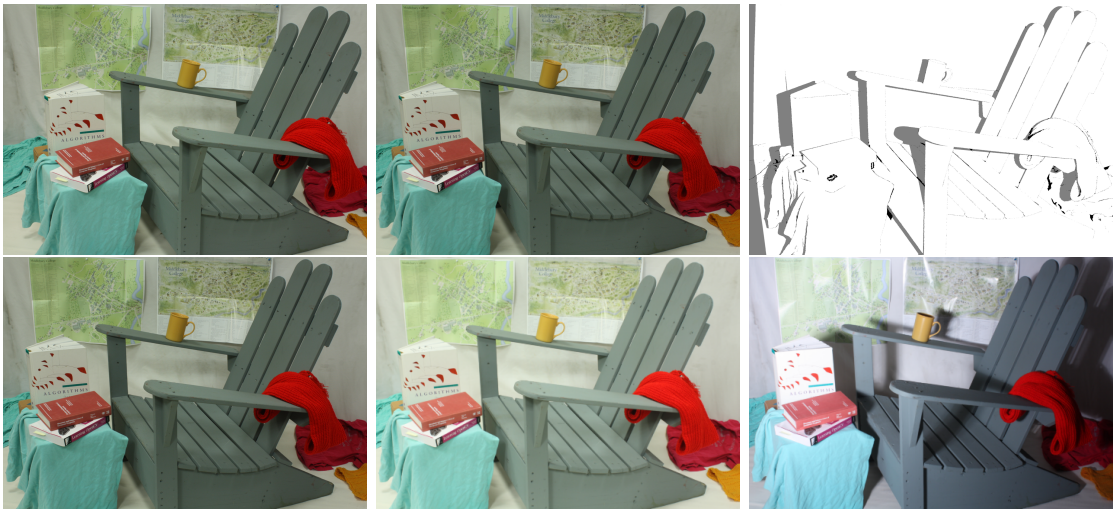


Figure 4.1: First row: The left and and right stereo image and the labels, where white means no occlusion, gray is occlusion and black are invalid pixels. Second row: Different light intensities taken from the right camera.

4.1.2 SceneFlow Dataset (Monkaa)

SceneFlow [69] is a synthetic datasets and provides 24 scenes containing 8688 stereo image pairs with left-to-right and right-to-left disparity maps (Figure 4.2). A scene is captured by two cameras which move around. The image size of all images is 540×950.

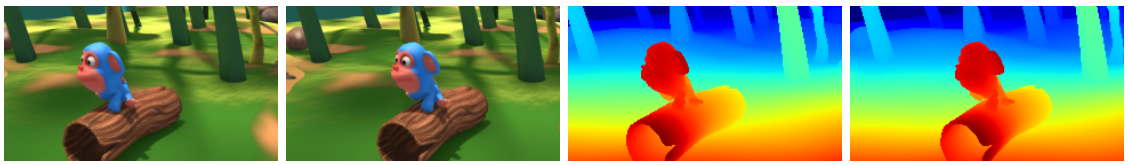


Figure 4.2: Monkaa example. A stereo image pair of scene with its disparity maps.

4.2 Data Pre-proccessing

Data pre-processing is a technique that transforms raw image data from a camera into a better understandable format for the models. The datasets needs to be split into training set and validation set, so that cross validation can be used to ensure the quality of the model on unseen data. Before that, we also have to resize the images and their disparity

maps to overcome the small available GPU memory. We compare in Figure 4.4 the results of resizing images with Gaussian pre-filtering.

4.2.1 Standardization

Standardization is a common used pre-processing technique compromised of mean subtraction and scaling by the standard deviation. If we subtract the mean on the input images, the data is centered around the origin. To normalize the input data we determine the unit variance, that is calculated by the mean μ divided by standard deviation σ per image per color channel:

$$\boldsymbol{X}^{(i)\prime} = \frac{\boldsymbol{X}^{(i)} - \boldsymbol{\mu}^{(i)}}{\boldsymbol{\sigma}^{(i)}}, \tag{4.1}$$

where $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_3]^T$ and $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \sigma_3]^T$ are 3×1 column vectors. For examining the zero mean, we can sum up the subtraction of the input data by the mean to 0:

$$\boldsymbol{X}^{(i)} - \boldsymbol{\mu}^{(i)} \stackrel{!}{=} 0. \tag{4.2}$$

For examining the unit variance, we can sum up the unit variance in Equation (4.1) to 1:

$$\sigma\big(\boldsymbol{X}^{(i)\prime}\big) \stackrel{!}{=} 1. \tag{4.3}$$

The mean subtraction reduces brightness variations and the division by the standard deviation mitigates variations in the spread of the data about the mean so that the two images have similar means and standard deviations. This is useful to compare similar images. The normalization is applied after symmetrically padding (Figure 2.5) the input images, otherwise the constraints Equation (4.2) and Equation (4.3) would not hold.

4.2.2 Cross Validation

Cross validation is a validation technique [76] to assess how the model generalize to independent data. Due to the fact of the small number of training samples in the training set, we decided to use 4-fold cross validation which splits the dataset in 4 equal sized sets. 3 sets are combined to use for the training set and 1 set is used for validation. The model is trained 4 times with 4 times different training and validation sets.

4.2.3 Data Augmentation - Random Crop

Goodfellow et al. [28] formulated data augmentation as generating more training data from existing ones. Beside geometric transformation, such as flipping, rotating, shearing, there also exists random cropping that does not violate geometric constraints. Due to the problem of a small dataset more data is needed to train the network, otherwise it overfits very quickly. For random crop, a region at the original image is cropped out. For

supervised learning, a new ground-truth must be created to the cropped out regions. In our case, the ground-truth is the occlusion map, which can be calculated with the *LRC* (Section 2.6.1) of the disparity maps provided in the Middlebury dataset.

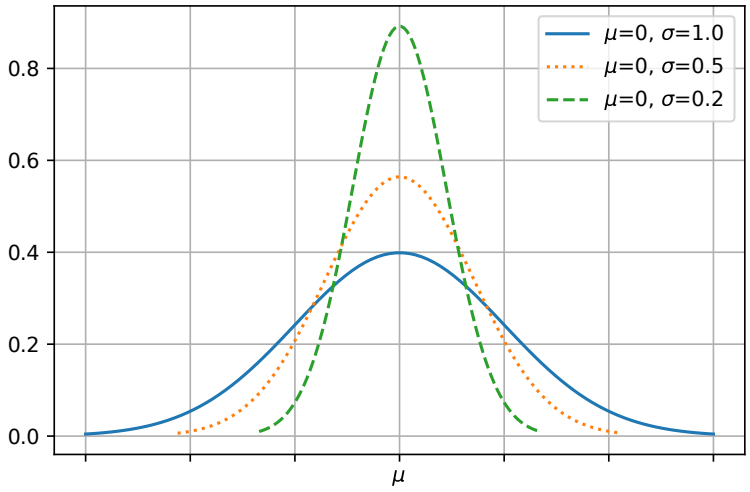


Figure 4.3: Gaussian filter examples.

4.2.4 Image Resizing

Image resizing is necessary to not overcome the *GPU* memory constraint and hence the data set images are downsampled. That is done by using a Gaussian filter, that promotes the homogeneous areas and more natural edges. The Gaussian filter is applied before interpolating the image with the nearest-neighbor method. First of all, the standard deviations σ for 2 dimensions must be calculated, that is accomplished by taking the scaling factors κ_i for the x and y-axis of the Gaussian filter

$$\sigma_i = \frac{\frac{1}{\kappa_i}}{3}, \qquad \kappa_i \in \mathbb{R} : i \in \{1, 2\},\ 0 < \kappa_i < 1,$$

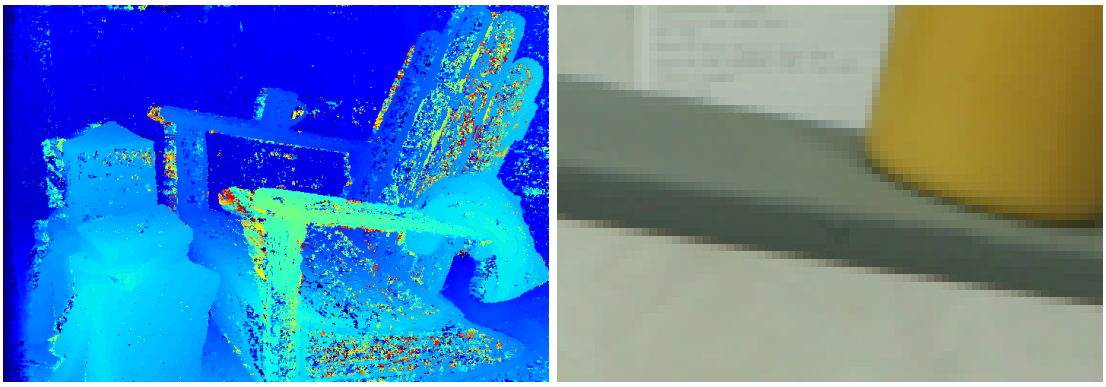
where the scaling factor is additionally divided by 3 to ensure that the output is not too blurry. Apply Gaussian filter separately for each channel of the image:

$$g(x)_i = \frac{1}{\sqrt{2\pi} \cdot \sigma_i} \cdot e^{-\frac{x^2}{2\sigma_i^2}},$$

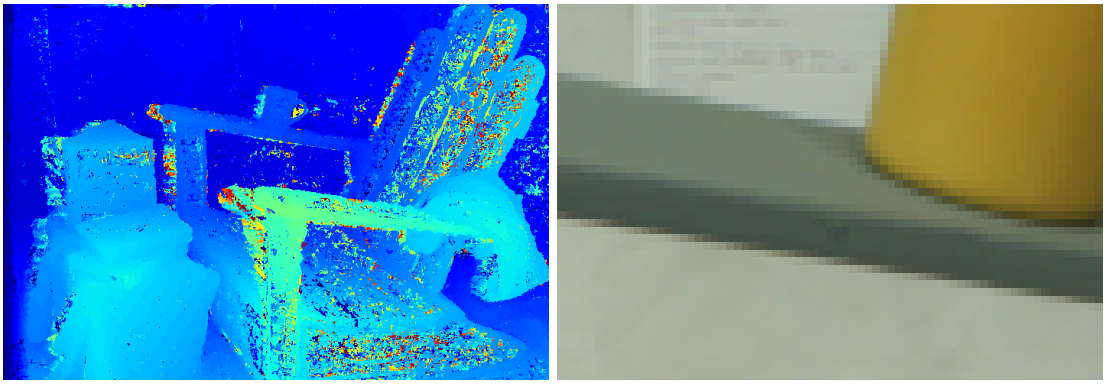
where x is the input sample. The Gaussian filter simulates the optical blur, that is more natural for the human vision system. Another advantage of the produced blur by the Gaussian filter (Figure 4.3) is shown in Figure 4.4, where the regions are more homogeneous, when we compute the stereo correlation followed by Section 2.5.3 to determine the disparity. Then, the values are interpolated per channel by the first-degree spline function.

4.2.5 Disparity Map Resizing

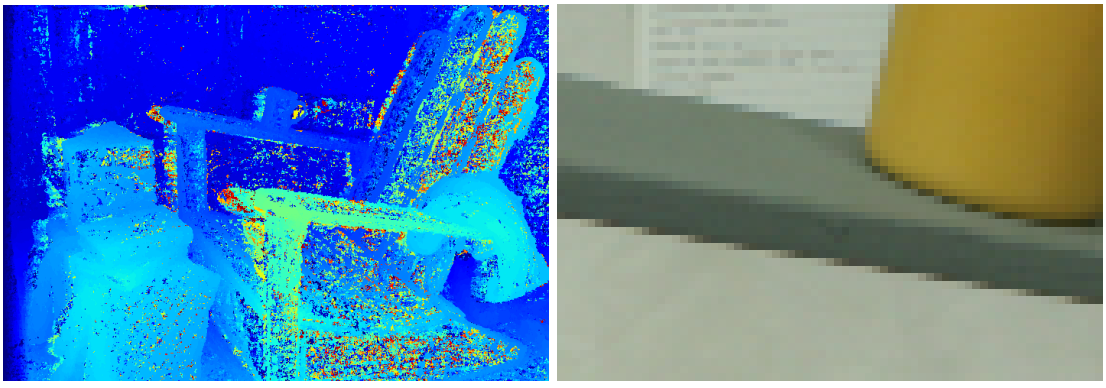
Disparity maps (e.g. Figure 1.3) are differently to normal images because disparities are not color intensities. We only used the nearest neighbors interpolation to resize the disparity map because it only copies the disparity values instead of interpolating them.



(a) Estimated disparity map and original image patch.



(b) Estimated disparity map and resized image patch with the first-degree spline function and Gaussian pre-filtering.



(c) Estimated disparity map and resized image patch without pre-filtering.

Figure 4.4: The original image is compared to other scaling methods, while the Gaussian pre-filtering achieves better results on homogeneous areas without pre-filtering.

4.3 Performance Metrics

In order to evaluate results achieved with the proposed neural networks, we use metrics to compare the similarity from the prediction with the ground truth. One is the Receiver Operating Characteristic (ROC) curve to find a threshold to classify pixels, whether a pixel occluded or not. The measurement Receiver Operating Characteristic (ROC) [75] that is widely used. However, this metric might be poor for images with a lot of background and only small areas are to detect. Therefore, we also investigate within the detected area the correct prediction.

4.3.1 Overall Accuracy

We have as output occluded and non-occluded pixels that we can compare with ground-truth labels. Finally, a valid mask $\mathbf{M}_{i,j}$ (Equation (3.2)) needs to be computed, that filters out the invalid pixels, which are marked as black in Figure 4.1, where the valid mask $\mathbf{M}_{i,j}$ is multiplied to the outcome of {TN, FN, TP, FP} element-wise, so that $\mathbf{M}_{i,j} = 0$ sets the pixels to zero and are not taken into account to the accuracy, divided the number of all possible pixels. The accuracy after thresholding the image can be calculated by

$$acc = \frac{\sum_{i,j} \mathbf{M}_{i,j}(1 - |\hat{\mathbf{Y}}_{i,j} - \mathbf{Y}_{i,j}|)}{\sum_{i,j} \mathbf{M}_{i,j}}, \tag{4.4}$$

where the inner part of the numerator is listed in Table 3.5 and is 1 *iff* $\tilde{\mathbf{Y}}_{i,j}$ and $\mathbf{Y}_{i,j}$ is equal. The valid mask $\mathbf{M}_{i,j}$ masks out the invalid pixels divided by all valid pixels.

4.3.2 Area Under Curve

Area Under Curve (AUC) tells us about the accuracy of the model. The accuracy depends on how well the model separates the group of non-occluded and occluded pixels. An area of 1 represents a perfect model and 0.5 represents a worthless model. The model can be classified as a traditional point system [77] {.90-1 = excellent (A), .80-.90 = good (B), .70-.80 = fair (C), .60-.70 = poor (D), .50-.60 = fail (F)}.

4.3.3 Correct predicted Pixels within occluded and non-occluded Areas

The accuracy over the whole segmented image is not sufficient. We want to distinguish between occluded and non-occluded areas and introduce two metrics which are capable to give the percentage of correct predicted pixels. The accuracy of occluded areas is calculated by

$$occ = \frac{\sum_{i,j} \mathbf{M}_{i,j} \cdot \hat{\mathbf{Y}}_{i,j} \cdot \mathbf{Y}_{i,j}}{\sum_{i,j} \mathbf{M}_{i,j} \cdot \mathbf{Y}_{i,j}}, \tag{4.5}$$

where only occluded pixels with the ground-truth $\mathbf{Y}$ is compared, divided by all available occluded pixels. In reverse, the accuracy of non-occluded areas is calculated

$$noc = \frac{\sum_{i,j} \mathbf{M}_{i,j} \cdot |1 - \hat{\mathbf{Y}}_{i,j}| \cdot |1 - \mathbf{Y}_{i,j}|}{\sum_{i,j} \mathbf{M}_{i,j} \cdot |1 - \mathbf{Y}_{i,j}|}, \tag{4.6}$$

where only occluded pixels of the ground-truth with the prediction is compared, divided by all available non-occluded pixels.

4.4 Detection Results

In this section, we discuss the detection results of our 4 models (see Section 3.1): "Direct", "Cor+2D", "Cor+3D". Both direct method and stereo correlation based methods have their characteristics. The stereo correlation has also limitations with not rectified input data because the correlation is only calculated with values in the same row of the other image. If the image is not (correctly) rectified we would not have the corresponding image points in the same row and hence this method has its difficulty to find its corresponding pixel. The direct method uses a 3×3 instead of a 1×3 kernel size that takes pixels in the vertical into account.

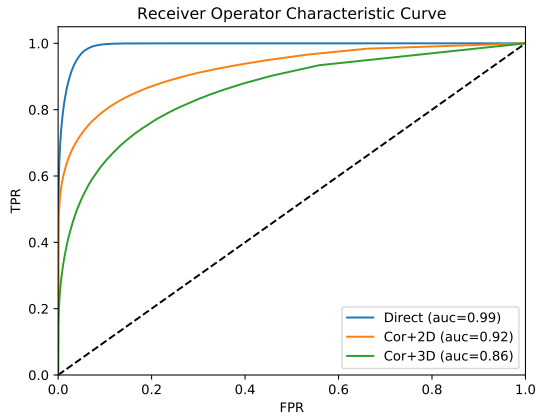
We have trained our model on the image size of 1/6 of the original image size and marked it as *MB-6*. We also evaluated our model with normal illumination and with different illuminations (see Figure 4.1).

In Table 4.2 we have summed up our results of our four methods with different datasets, where MB stands for the Middlebury dataset and SF stands for the Sceneflow dataset. MB Normal Light refers that the methods are trained with only images with the same illumination and restricts the MB dataset size to 34 training image and 12 validation images. Due to the small training set size we have problems with overfitting except with "Cor+2D" (Section 2.2.3.3), that has also achieved best results. MB Light Differ refers to the additional images with different illuminations. The training set size expands to 102 images and the validation set size to 36 images. The validation results are remained basically the same, although the different illuminations constitute a more difficult training scenario.

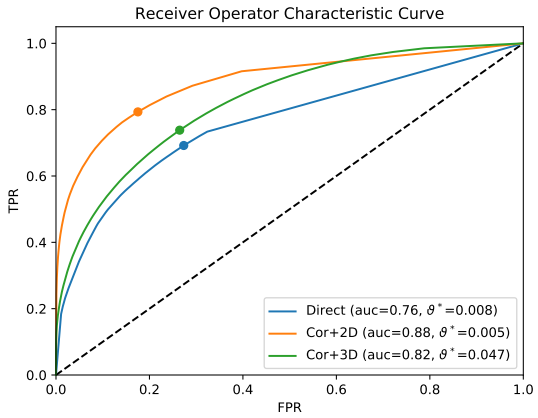
We used cross-validation to assess how the results will generalize on independent data. To measure the performance, we use a 4-fold to cross-validate our models. A 4-fold compared to a 3-fold has the advantage that the training set is a little bit larger. We use following performance measures: the accuracy Equation (4.4), the AUC, the percentage of correct predicted occluded pixels *occ* Equation (4.5) and the percentage of correct predicted non-occluded pixels *noc* Equation (4.6).

In order to improve the results, we pre-trained our methods with SF and fine-tuned them with MB. We have used a batch size of 3 and fine-tuned it with a batch size of 1. SF is a large dataset that provides 7153 stereo image pairs for training set and 1503 stereo

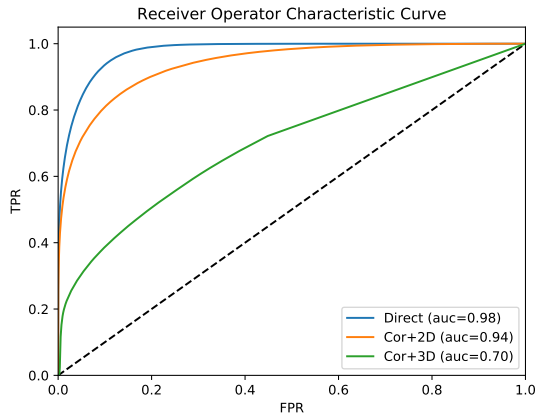
image pairs for validation. This helps to get rid of our overfitting problem because of the small Middlebury dataset. This can be observed by the results of the direct method. The last two rows in Table 4.2 show that the accuracy on the validation set is only about 85%. In Table 4.2, we have evaluated every k-fold of MB-6 and MB-Q if the occlusion detection is completely non-occluded. The accuracy is still about around 82%. The good is that we do not have to use random crop anymore, which shows a little bit worse accuracy on the validation set when we only train "Cor+3D" with MB. Hence, the results on the validation have been improved.



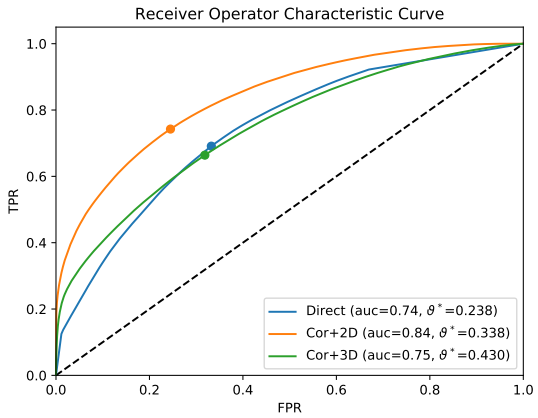
(a) Training set. Normal illumination.



(b) Validation set. Normal illumination.



(c) Training set. Different illumination.



(d) Validation set. Different illumination.

Figure 4.5: *ROC* plots from training and validation set with normal illumination and different illumination.

It is difficult to evaluate the detection results with only the *ROC* plot because of the low percentage of occluded pixels within a stereo pair. Downsizing from the original image size to only 1/6 or quarter can have little change of occluded pixels. So, we have evaluated the number of percentage of every k-fold in Table 4.1. The percentage of the

Occlusions				
k-fold	MB-6		MB-Q	
	train	val	train	val
fold 1	17.25	16.28	16.42	15.59
fold 2	15.54	21.11	14.82	20.14
fold 3	17.79	14.74	16.98	14.01
fold 4	17.11	16.49	16.40	15.65
avg	16.92	17.16	16.16	16.35
all	16.99		16.21	

Table 4.1: Percentage of occlusions of every k-fold for the Middlebury dataset of 1/6 and quarter size.

occlusions is between 14 and 21 percent per k-fold (Table 4.1). For illustration, we have plotted the ROC curve in Figure 4.5 to illustrate additionally the performance of all 4 models. In Figure 4.5a, the direct method (blue curve) reaches almost the left-upper corner, that indicates an AUC of almost 1.0 (highest score). Unfortunately, this is on the training set and shows us a clear overfitting. On the validation set (Figure 4.5b), the direct method achieves an AUC only of 0.758. The best method is "Cor+2D" (orange curve) that generalizes best on both Normal Light and Light Differ. In the validation set (see Figure 4.9) the method "Cor+2D" also shows good visual results and performs best in overall compared to other methods. We have marked the optimal threshold ϑ^* on the ROC curve with a point.

In Figure 4.6, Figure 4.8, Figure 4.8 and Figure 4.9, we have some example occlusion maps from the MB dataset of each method that are before and after applying the global threshold. The direct method shows in Figure 4.6 the overfitting on the training set, what would be perfect results. On the validation set it shows some correct predicted occluded pixels while there are many wrong predicted ones. Pre-training the direct method helped to perform better, but still there is a lack in the final results. In Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13, we have also some examples of the occlusion maps of the SF dataset. As marked bold face in Table 4.2 (last three rows) the method "Cor+2D" results best. The method "Cor+3D" does have performance issues. It is difficult to train it because of the large number of learnable parameters. So we use random crop as data augmentation technique and could achieve better prediction results. The results of "Cor+3D" show a lot of wrong predicted in occluded pixels (see Figure 4.9) as well as in homogeneous areas.

Methods		Experiments												
		training set ($\mu \pm \sigma$ in percent)				validation set ($\mu \pm \sigma$ in percent)				Batch Size	Rand. Bat.	Loss	Image Size	#Params (in Mio.)
		acc	auc	occ	noc	acc	auc	occ	noc					
MB-6	fold 1	81.95	50	0	1	83.49	50	0	1				6	
	fold 2	83.80	50	0	1	78.23	50	0	1				6	
	fold 3	81.68	50	0	1	84.23	50	0	1				6	
	fold 4	82.23	50	0	1	82.87	50	0	1				6	
	all	82.35	50	0	1								6	
MB-Q	fold 1	82.79	50	0	1	84.16	50	0	1				Q	
	fold 2	84.54	50	0	1	79.20	50	0	1				Q	
	fold 3	82.49	50	0	1	85.01	50	0	1				Q	
	fold 4	82.95	50	0	1	83.71	50	0	1				Q	
	all	83.15	50	0	1								Q	
MB	Direct	86.3 ± 6.3	97.6 ± 2.5	97.6 ± 3.4	83.8 ± 7.3	74.8 ± 8.1	72.3 ± 5.4	60.1 ± 6.0	73.9 ± 9.4	1		CE	6	0.35
Normal	Cor+2D	78.1 ± 8.1	91.9 ± 1.4	88.2 ± 3.9	75.1 ± 9.9	$\mathbf{81.9} \pm 6.2$	85.6 ± 1.9	74.5 ± 4.6	83.1 ± 8.7	1		CE	6	0.58
Light	Cor+3D	82.7 ± 4.9	84.7 ± 1.4	61.2 ± 9.6	89.5 ± 4.5	79.3 ± 5.1	81.1 ± 2.5	70.7 ± 4.9	76.0 ± 3.1	4	✓	CE	6	1.1
MB	Direct	85.8 ± 6.3	96.8 ± 1.1	95.1 ± 3.4	83.7 ± 8.4	69.5 ± 3.3	72.9 ± 3.2	62.0 ± 9.2	71.8 ± 5.0	1		CE	6	0.35
Light	Cor+2D	85.1 ± 3.2	93.6 ± 0.3	85.0 ± 4.6	85.1 ± 4.8	79.5 ± 2.3	83.8 ± 1.5	66.8 ± 6.4	82.5 ± 4.1	1		CE	6	0.58
Differ	Cor+3D	84.7 ± 0.7	75.9 ± 15.3	19.6 ± 1.6	98.6 ± 0.1	$\mathbf{80.4} \pm 4.7$	81.7 ± 6.2	55.7 ± 3.8	85.4 ± 5.0	4	✓	CE	6	1.1
SF-MB	Direct	83.3 ± 7.4	96.9 ± 1.3	85.7 ± 17.2	77.9 ± 10.1	77.6 ± 7.2	82.6 ± 9.5	84.3 ± 12.8	81.0 ± 5.7	1		CE	6	0.35
Normal	Cor+2D	83.6 ± 3.5	92.2 ± 2.7	83.8 ± 8.8	83.3 ± 6.3	81.6 ± 5.2	89.8 ± 2.1	82.9 ± 9.8	82.9 ± 5.4	1		CE	6	0.58
Light	Cor+3D	83.0 ± 1.6	91.1 ± 0.4	84.6 ± 8.4	81.1 ± 10.1	$\mathbf{83.7} \pm 3.2$	54.4 ± 3.4	66.5 ± 2.9	82.9 ± 1.4	1		CE	6	1.1
SF-MB	Direct	84.9 ± 2.6	92.3 ± 4.7	83.2 ± 10.1	85.1 ± 1.3	79.8 ± 6.7	84.5 ± 1.7	82.1 ± 7.4	79.3 ± 6.6	1		CE	6	0.35
Light	Cor+2D	84.1 ± 0.9	90.2 ± 0.9	75.4 ± 3.9	86.1 ± 1.9	$\mathbf{82.3} \pm 1.5$	86.6 ± 1.1	69.3 ± 2.1	69.7 ± 3.8	1		CE	6	0.58
Differ	Cor+3D	81.9 ± 1.1	83.6 ± 3.0	65.6 ± 15.0	81.4 ± 8.5	81.4 ± 1.1	77.9 ± 1.1	55.7 ± 7.5	81.1 ± 6.0	1		CE	6	1.1
SF	Direct	91.1 ± 1.8	98.7 ± 0.9	97.7 ± 1.8	89.8 ± 2.1	80.3 ± 3.8	84.0 ± 5.1	67.2 ± 14.3	84.7 ± 4.3	1		CE	6	0.35
	Cor+2D	93.4 ± 2.7	99.7 ± 0.4	97.5 ± 4.3	92.8 ± 6.1	$\mathbf{85.4} \pm 6.3$	84.2 ± 9.5	72.0 ± 8.6	87.7 ± 5.7	1		CE	6	0.58
	Cor+3D	85.2 ± 3.2	89.0 ± 1.9	84.3 ± 9.6	84.7 ± 3.7	82.9 ± 2.8	90.6 ± 3.7	56.1 ± 7.9	85.6 ± 5.0	1		CE	6	1.1

Table 4.2: Experiments of direct method and of stereo correlation with global thresholding.

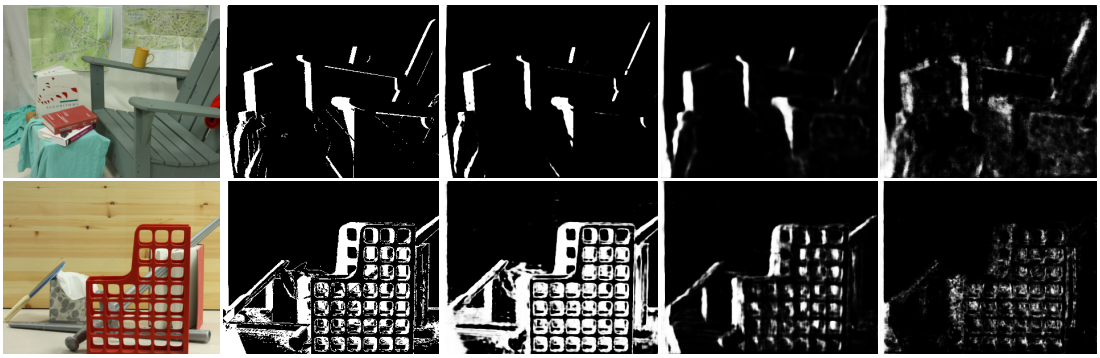


Figure 4.6: MB training set: The soft prediction of selected images: Adirondack and Sword2.

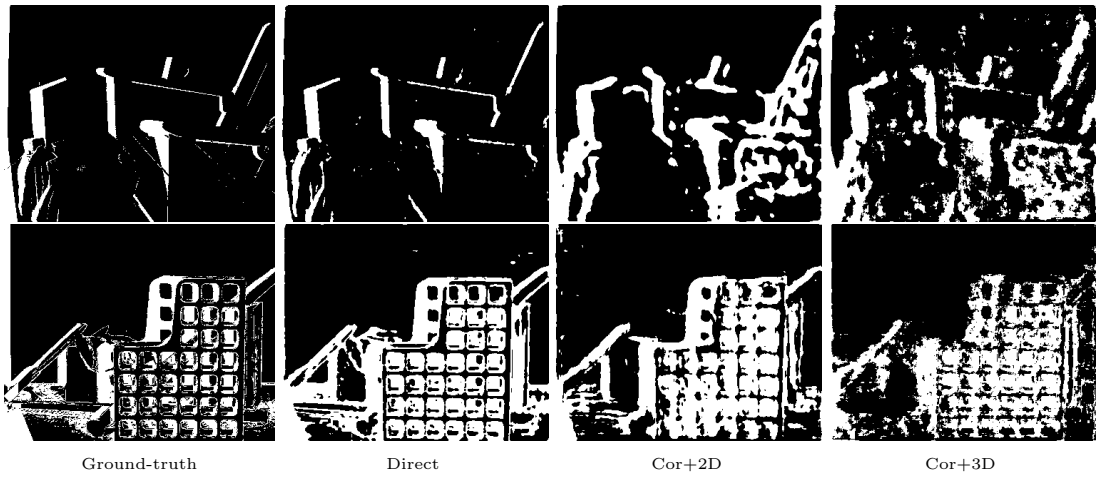


Figure 4.7: MB training set: The soft predictions are classified by the optimal threshold.

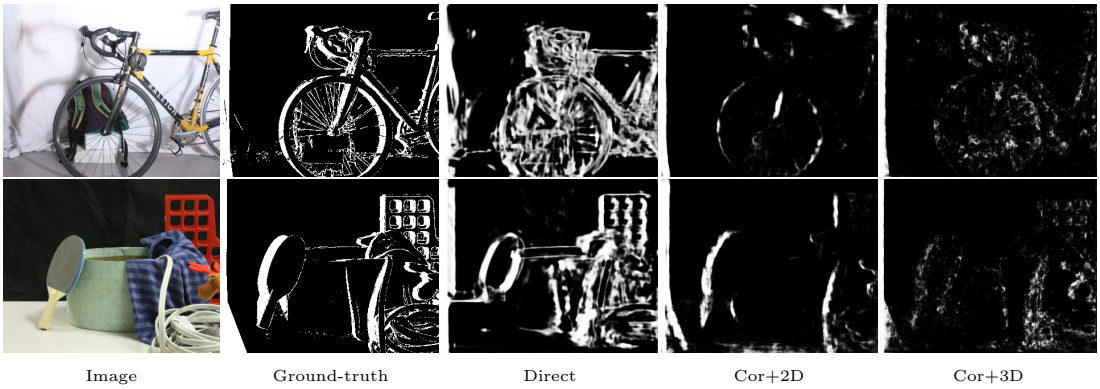


Figure 4.8: MB validation set: The soft prediction of selected images: Bicycle1 and Cable.

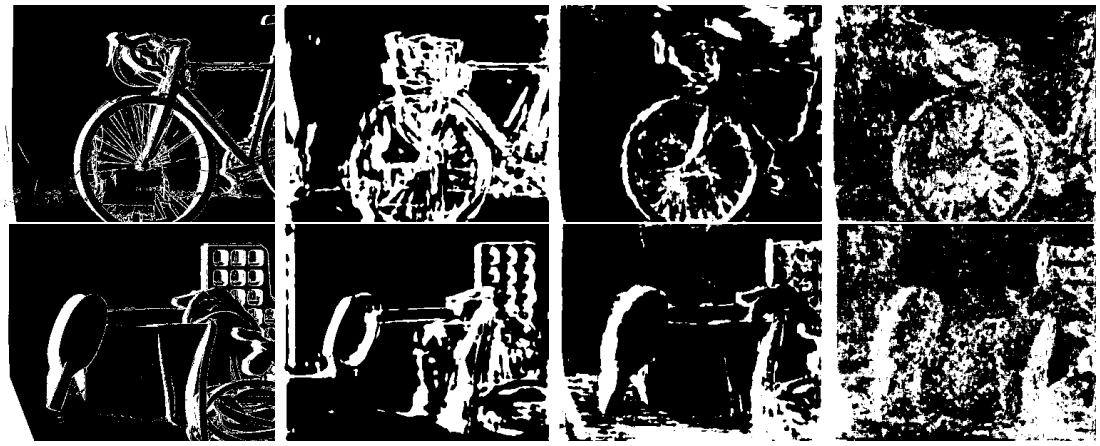


Figure 4.9: MB validation set: The soft predictions are classified by the optimal threshold.

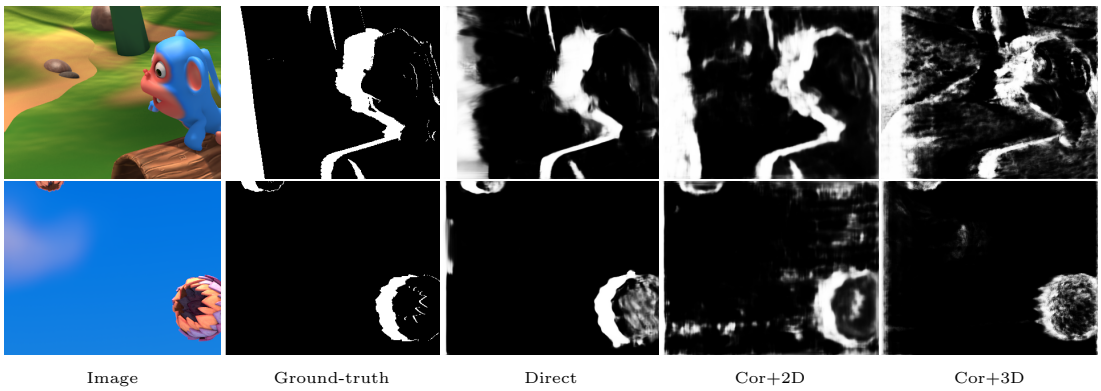


Figure 4.10: SF training set: The soft prediction of selected images.

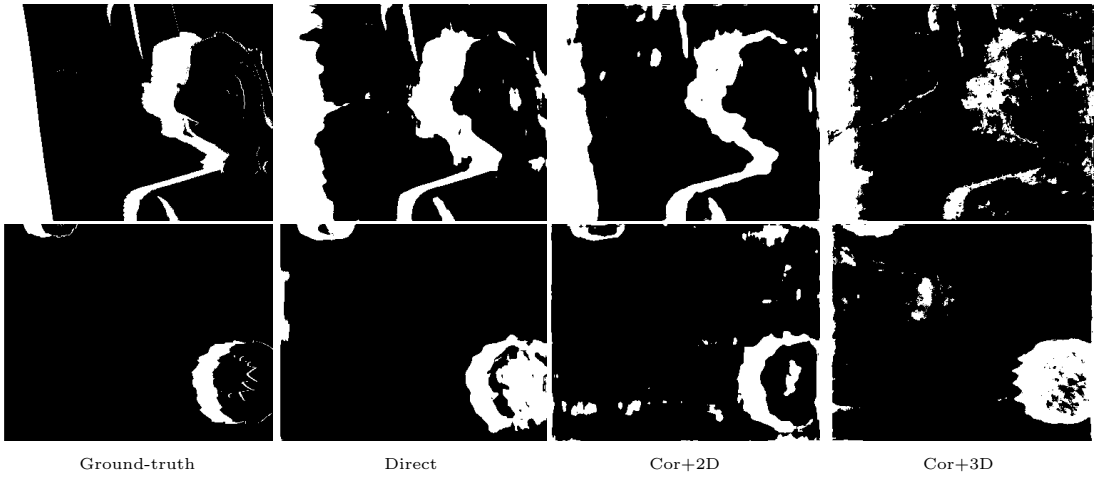


Figure 4.11: SF training set: The soft predictions are classified by the optimal threshold.

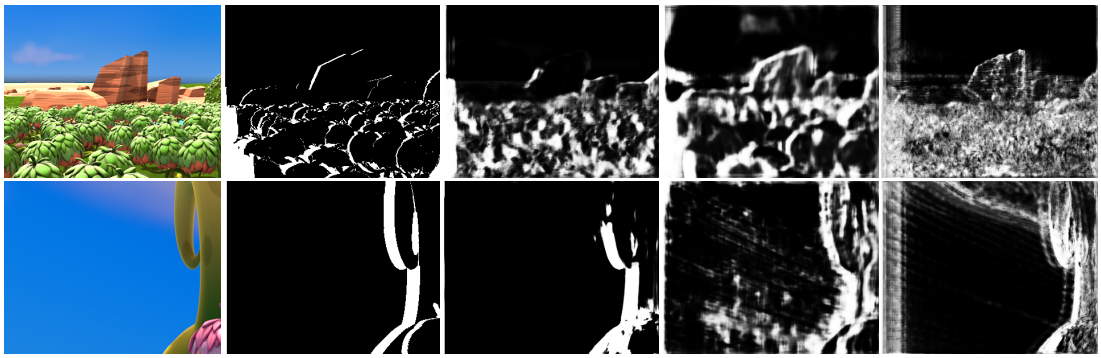


Figure 4.12: SF validation set: The soft prediction of selected images.

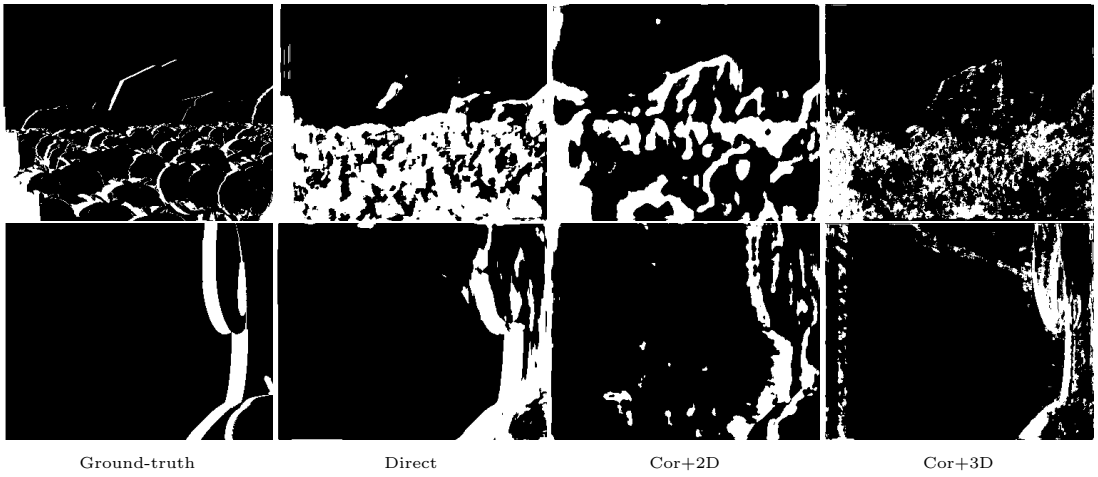


Figure 4.13: SF validation set: The soft predictions are classified by the optimal threshold.

5 Conclusion and Outlook

Contents

In this chapter, we discuss the limitations of the methods we have investigated and discuss possible improvements. In addition, we will suggest state-of-the-art methods to adapt their model to the occlusion problem.

5.1 Conclusion

In this thesis, we have discussed several supervised *CNN* methods to detect pixel-wise occlusions. The "direct method" differs from the other methods as the occlusions are not learned via stereo correlation computation of a stereo image pair. We concatenated the output of both unary *CNN*s. The concatenation is used as input of a 2D convolutional layer to calculate the deviation of both input features.

The other three methods are based on the stereo correlation. Therefore, we have transferred a CUDA/C++ implementation to tensorflow. We checked the correctness by numerically approximating the gradient of the stereo correlation with forward difference. The CUDA implementation offers pre-trained unary *CNN*s with 3 and with 7 layers. The computation with 7 layers is more accurate and we have decided to take it for our methods. The method that refines the stereo correlation with dilated 2D convolutions gives the best results. The last method refine the stereo correlation with a 3D convolution as aggregation. It also uses the Cross-Entropy (*CE*) as loss function.

We trained and evaluated our methods with the Middlebury dataset. Since this dataset is small, we pretrained our methods with the large synthetic dataset Sceneflow (Monkaa).

5.2 Future Work

As future work, we want to replace the aggregation for the direct method and the stereo correlation method with a 3D convolution hour-glass architecture. Since the input consists concatenated RGB-images in the SymmNet, we would have the concatenated output of the unary *CNN*s of the direct method or the cost volume of the stereo correlation. There have been already proposed 3D-hourglass architectures for stereo matching. We suggest to adapt them to occlusion detection.

One approach is introduced by Chang and Chen [78] who proposed a Pyramid Stereo Matching Network (PSMNet) to compute disparities from stereo images. 3D convolutions are used in several hourglass architecture pipelines to refine the cost volume. Another approach is introduced by Alex Kendall et al. [79] who proposed only one big hour-glass architecture. The hour-glass architectures use downsampled input features from other previous layers and add/concatenate them in the upsampling layers. The output of the hour-glasses is combined via regression that we would be interesting to be changed to classification.

For these experiments, we would suggest to alter the current Cross-Entropy (*CE*). The current *CE* only treats the occlusions in the left image, but not in the right image. In Section 2.4.1 we described the two-sided *CE* and stated that computing both occlusion maps simultaneously would help for more accurate results.

Last but not least, our models are only designed for supervised learning. We have labeled data and our models can only learn from this. An unsupervised approach or a reinforcement approach would broaden the applicability of this problem. The model should find patterns on its own instead of relying on labeled data.

A

List of Acronyms

$\bar{o}$	non-occluded
o	occluded
Adam	Adaptive Moment Estimation
AUC	Area Under Curve
CE	Cross-Entropy
CNN	Convolutional Neural Network
CRF	Conditional Random Fields
CT	Census Transform
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GPU	Graphics Processing Unit
HL	Hinge Loss
iff	if-and-only-if
LRC	Left-Right-Cross-Checking
MLP	Multilayer Perceptron
PR	Precision-Recall
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SGM	Semi Global Matching
SSD	Sum of Squared Differences
TN	True Negative
TP	True Positive
TPR	True Positive Rate
UAT	Universal Approximator Theorem

Bibliography

[1] A. Ortiz, "Illustrates how Image Rectification simplifies the search space in Stereo Correlation Matching," 2008. (page xix, 2)

[2] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision"*. Cambridge University Press, second ed., 2004. (page xix, 3)

[3] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang, and P. Westling, "High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth," in *German Conference on Pattern Recognition (GCPR)*, 2014. (page xix, 4, 31, 42, 57)

[4] David Borland and Taylor, Russell M., "Rainbow Color Map (still) considered harmful," *IEEE Computer Graphics and Applications*, vol. 27, no. 2, pp. 14–17, 2007. (page xix, 4)

[5] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 4489–4497, 2015. (page xix, 17)

[6] J. Žbontar and Y. LeCun, "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches," *Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, pp. 2287–2318, 2016. (page xix, 30, 31, 34, 36)

[7] K. Zhang, J. Lu, and G. Lafruit, "Cross-Based Local Stereo Matching Using Orthogonal Integral Images," *Circuits and Systems for Video Technology (TCSVT)*, vol. 19, no. 7, pp. 1073–1079, 2009. (page xix, 32, 33)

[8] P. Knöbelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock, "End-to-End Training of hybrid CNN-CRF Models for Stereo," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2017. (page xix, xxi, 4, 34, 35, 43, 47, 48)

[9] Y. Boykov and O. Veksler, "Graph Cuts in Vision and Graphics: Theories and Applications," in *Handbook of Mathematical Models in Computer Vision*, 2006. (page xx, 38)

[10] L. Ang and Y. Zejian, "SymmNet: A Symmetric Convolutional Neural Network for Occlusion Detection," in *Proceedings of British Machine Vision Conference (BMVC)*, 2018. (page xx, 4, 14, 39, 41, 47)

[11] Sharpr, "This is a Recreation of the File Receiver_Operating_Characteristic.png as a Vector Graphics Image in SVG Format.," 2015. (page xx, 53)

[12] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. (page 1)

[13] V. Kolmogorov and R. Zabih, "Computing visual Correspondences with Occlusions using Graph Cuts," *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 1–33, 2001. (page 4, 37)

[14] G. Egnal and R. P. Wildes, "Detecting Binocular Half-Occlusions: Empirical Comparisons of Five Approaches," *Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, no. 8, pp. 1127–1133, 2002. (page 4, 36)

[15] C. L. Zitnick and S. B. Kang, "Stereo for Image-Based Rendering using Image Over-Segmentation," *International Journal of Computer Vision*, vol. 75, pp. 49–65, Oct 2007. (page 7)

[16] R. Zabih and J. Woodfill, "Non-parametric local Transforms for computing Visual Correspondence," in *Proceedings of European Conference on Computer Vision (ECCV)*, pp. 151–158, Springer-Verlag, 1994. (page 8)

[17] L. Ma, J. Li, J. Ma, and H. Zhang, "A Modified Census Transform Based on the Neighborhood Information for Stereo Matching Algorithm.," pp. 533–538, IEEE Computer Society, 2013. (page 8)

[18] S. Meister, J. Hur, and S. Roth, "UnFlow: Unsupervised Learning of Optical Flow with a Bidirectional Census Loss," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2017. (page 9)

[19] D. H. Hubel and T. N. Wiesel, "Receptive Fields of Single Neurons in the Cat's Striate Cortex," *Journal of Physiology*, vol. 148, pp. 574–591, 1959. (page 10)

[20] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, pp. 65–386, 1958. (page 10)

[21] K. Hornik, "Approximation Capabilities of Multilayer Feedforward Networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. (page 11)

[22] B. C. Csáji, "Approximation with Artificial Neural Networks," Master's thesis, Eotvos Lorand University (ELTE), 2001. (page 11)

[23] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *International Conference on Machine Learning (ICML)*, International Conference on Machine Learning (ICML), (USA), pp. 807–814, Omnipress, 2010. (page 12)

[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities improve Neural Network Acoustic Models," in *International Conference on Machine Learning (ICML)*, 2013. (page 13)

[25] M. Nielsen, "Neural Networks and Deep Learning," *Determination Press*, 2015. (page 13, 19, 20, 24, 30)

[26] Y. LeCun and C. Cortes, "MNIST handwritten Digit Database," 2010. (page 13)

[27] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object Recognition with Gradient-Based Learning," in *Shape, Contour and Grouping in Computer Vision*, (London, UK), pp. 319–, Springer-Verlag, 1999. (page 14)

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. (page 14, 15, 20, 59)

[29] A. Martin *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," 2015. (page 15, 16, 49, 57)

[30] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," *Neural Information Processing Systems (NIPS)*, vol. abs/1701.04128, 2017. (page 15)

[31] V. Dumoulin and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning," *arXiv*, vol. abs/1603.07285, 2016. (page 15)

[32] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," *International Conference for Learning Representations (ICLR)*, vol. abs/1511.07122, 2015. (page 16)

[33] R. Hou, C. Chen, and M. Shah, "An end-to-end 3D Convolutional Neural Network for Action Detection and Segmentation in Videos," *arXiv*, vol. abs/1712.01111, 2017. (page 17)

[34] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning Back Propagation," tech. rep., Carnegie–Mellon University, 1986. (page 21)

[35] B. Polyak, "Some Methods of Speeding Up the Convergence of Iteration Methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1 – 17, 1964. (page 21)

[36] R. A. Jacobs, "Increased Rates of Convergence through Learning Rate Adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295 – 307, 1988. (page 21)

[37] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2121–2159, 2011. (page 21)

[38] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," *arXiv*, vol. abs/1609.04747, 2016. (page 22)

[39] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference for Learning Representations (ICLR)*, vol. abs/1412.6980, 2014. (page 22, 57)

[40] A. Y. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "UFLDL Tutorial." `http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork`, 2013. (page 24, 25)

[41] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies," in *Field Guide to Dynamical Recurrent Networks*, IEEE Press, 2001. (page 27)

[42] "Untersuchungen zu dynamischen Neuronalen Netzen," Master's thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991. (page 27)

[43] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Information Processing Systems (NIPS)*, (London, UK), pp. 9–50, Springer-Verlag, 1998. (page 27, 28)

[44] R. Pascanu, T. Mikolov, and Y. Bengio, "On the Difficulty of training Recurrent Neural Networks," *International Conference on Machine Learning (ICML)*. (page 27)

[45] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256, 2010. (page 28, 29)

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1512.03385, 2015. (page 29)

[47] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance Introduction to Machine Learning for Physicists," *arXiv*, p. arXiv:1803.08823, 2018. (page 29)

[48] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995. (page 30)

[49] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996. (page 30)

[50] A. E. Hörl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. (page 30)

[51] H. Zou and T. Hastie, "Regularization and variable Selection via the Elastic Net," *Journal of the Royal Statistical Society, Series B*, vol. 67, pp. 301–320, 2005. (page 30)

[52] M. Menze, C. Heipke, and A. Geiger, "Joint 3D Estimation of Vehicles and Scene Flow," in *International Society for Photogrammetry and Remote Sensing (ISPRS)*, 2015. (page 31)

[53] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger, and R. Shah, "Signature Verification using a "Siamese" Time Delay Neural Network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, p. 25, 1993. (page 31)

[54] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification using a Siamese Time Delay Neural Network," in *Neural Information Processing Systems (NIPS)*, (San Francisco, CA, USA), pp. 737–744, Morgan Kaufmann Publishers Inc., 1993. (page 31)

[55] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On Building an accurate Stereo Matching System on Graphics Hardware," in *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 467–474, 2011. (page 32)

[56] H. Hirschmüller, "Accurate and efficient Stereo Processing by Semi-Global Matching and Mutual Information," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 807–814 vol. 2, 2005. (page 32, 33, 34)

[57] H. Hirschmüller, M. Buder, and I. Ernst, "Memory Efficient Semi-Global Matching," in *The Congress of the International Society for Photogrammetry and Remote Sensing*, 2012. (page 34)

[58] R. Spangenberg, T. Langner, and R. Rojas, "Weighted Semi-Global Matching and Center-Symmetric Census Transform for Robust Driver Assistance," in *Computer Analysis of Images and Patterns* (R. Wilson, E. Hancock, A. Bors, and W. Smith, eds.), pp. 34–41, Springer-Verlag, 2013. (page 34)

[59] J. Zbontar and Y. LeCun, "Stereo Matching by Training a Convolutional Neural Network to compare Image Patches," *Journal of Machine Learning Research (JMLR)*, vol. 17, pp. 1–32, 2016. (page 35)

[60] C. Bailer, K. Varanasi, and D. Stricker, "CNN based Patch Matching for Optical Flow with thresholded Hinge Loss," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1607.08064, 2016. (page 35)

[61] M. Brown, G. Hua, and S. Winder, "Discriminative Learning of Local Image Descriptors," *Pattern Analysis and Machine Intelligence (PAMI)*, vol. 33, no. 1, pp. 43–57, 2011. (page 35)

[62] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational Stereo," *Pattern Analysis and Machine Intelligence (PAMI)*, vol. 25, no. 8, pp. 993–1008, 2003. (page 36)

[63] H. Hirschmüller, P. R. Innocent, and J. Garibaldi, "Real-Time Correlation-Based Stereo Vision with reduced Border Errors," *International Journal of Computer Vision (IJCV)*, vol. 47, no. 1, pp. 229–246, 2002. (page 36)

[64] R. Trapp, S. Drüe, and G. Hartmann, "Stereo matching with implicit Detection of Occlusions," in *Proceedings of European Conference on Computer Vision (ECCV)* (H. Burkhardt and B. Neumann, eds.), pp. 17–33, Springer-Verlag, 1998. (page 36)

[65] P. Knöbelreiter, C. Vogel, and T. Pock, "Self-Supervised Learning for Stereo Reconstruction on Aerial Images," *International Society for Photogrammetry and Remote Sensing (ISPRS)*, 2018. (page 37)

[66] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. 1962. (page 37)

[67] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2017. (page 40)

[68] P. Lorenz and P. Roth, "Deep Learning Architectures for Estimating Optical Flow." `https://github.com/computeVision/optical_flow/blob/master/optical_flow.pdf`, 2018. (page 40)

[69] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A Large Dataset to train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134. (page 41, 58)

[70] H. Hirschmüller and D. Scharstein, "Evaluation of Cost Functions for Stereo Matching," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2007. (page 42)

[71] J. Kim, V. Kolmogorov, and R. Zabih, "Visual Correspondence Using Energy Minimization and Mutual Information," in *Proceedings of International Conference on Computer Vision (ICCV)*, (Washington, DC, USA), pp. 1033–, IEEE Computer Society, 2003. (page 42)

[72] F. Yu, V. Koltun, and T. A. Funkhouser, "Dilated Residual Networks," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1705.09914, 2017. (page 45)

[73] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic Differentiation in Machine Learning: a Survey," *Journal of Machine Learning Research*, vol. abs/1502.05767, 2015. (page 49)

[74] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast Cost-Volume Filtering for Visual Correspondence and Beyond," *Pattern Analysis and Machine Intelligence (PAMI)*, vol. 35, no. 2, pp. 504–511, 2013. (page 51)

[75] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006. ROC Analysis in Pattern Recognition. (page 52, 63)

[76] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R.* Springer-Verlag, 2014. (page 59)

[77] A. P. Bradley, "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms," *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. 30, no. 7, pp. 1145–1159, 1997. (page 63)

[78] J.-R. Chang and Y.-S. Chen, "Pyramid Stereo Matching Network," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2018. (page 72)

[79] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-End Learning of Geometry and Context for Deep Stereo Regression," *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 66–75, 2017. (page 72)