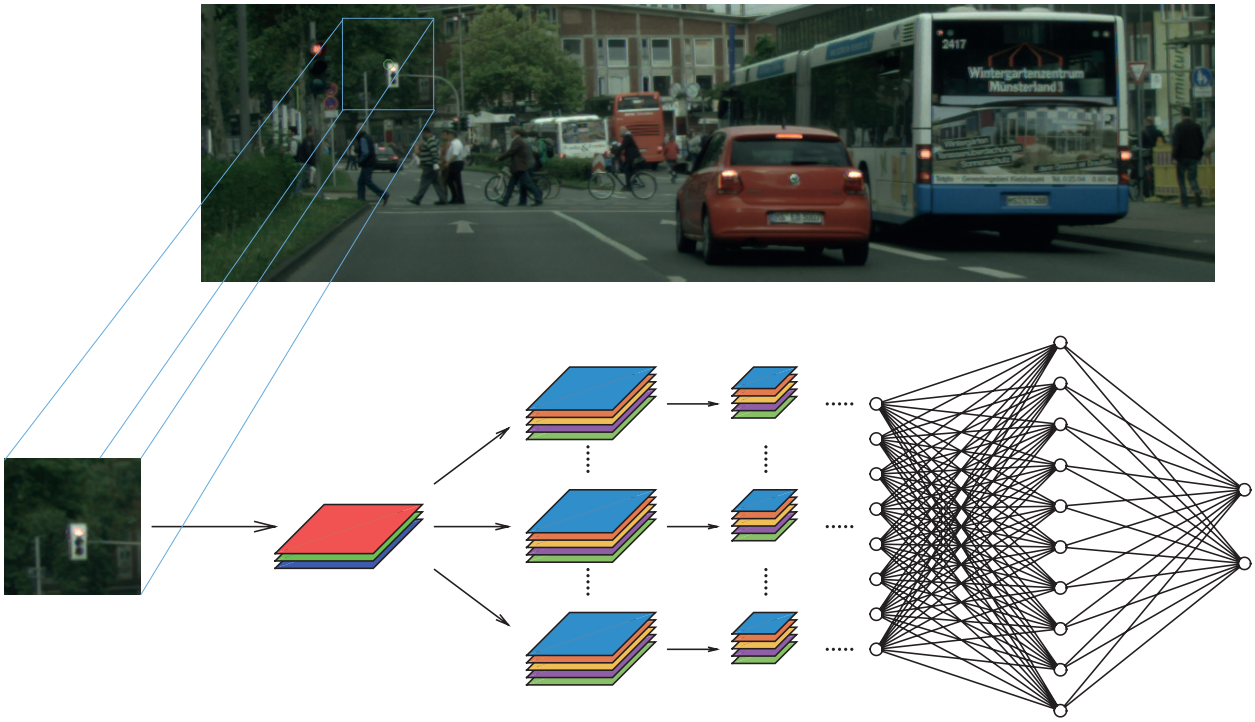
CHALMERS
UNIVERSITY OF TECHNOLOGY



Feature point description and classification for urban street scenes using convolutional neural networks

Master's thesis in Complex Adaptive Systems

HENRIK WALLENIUS

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:EX028

Feature point description and classification for urban street scenes using convolutional neural networks

HENRIK WALLENIUS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Image Analysis and Computer Vision
Chalmers University of Technology
Gothenburg, Sweden 2016

Feature point description and classification for urban street scenes using convolutional neural networks
HENRIK WALLENIUS

Supervisor: Erik Stenborg, Department of Signals and Systems
Examiner: Fredrik Kahl, Department of Signals and Systems

Master's Thesis 2016:EX028
Department of Signals and Systems
Image Analysis and Computer Vision
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The graphical representation of classifier. Given a detected feature point, a color image patch, is fed though the network, starting at a convolutional layer, followed by a max pooling layer and ending with a fully connected neural network.

Typeset in LaTeX
Gothenburg, Sweden 2016

Feature point description and classification for urban street scenes using convolutional neural networks.
HENRIK WALLENIUS
Department of Signals and Systems
Chalmers University of Technology

Abstract

The positioning problem may be solved by localizing features in a video stream and matching these with a map of features. However, this technique requires a unique representation of the features, which is not feasible using traditional methods.
In this thesis one method for a feature point descriptor and one for a feature point classifier, applicable in urban street scenes, using a convolutional neural network approach are proposed. The thesis presents the concept of training a feature descriptor to be invariant with respect to scale, rotation and noise given a training set of image patches modified by these transformations. The feature point classifier is trained given a predefined partition of stable and unstable classes, where a class is considered stable if it is expected to always appear at the same location in a scene, e.g. a building, and unstable otherwise, e.g. a pedestrian.
The results from the trained descriptor were compared with the SIFT descriptor, on urban street scenes, and were found to perform better by comparing their receiver operating characteristics. The results from the trained feature point classifier were compared with a semantic segmentation technique and yield similar performance on the entire validation set and significant improvements on small objects.

Keywords: computer vision, feature description, feature detection, convolutional neural networks, neural networks, urban street scenes.

Acknowledgements

Firstly, I would like to thank my supervisor Erik Stenborg and my examiner Fredrik Kahl for valuable discussions, support and ideas throughout the work. Furthermore, I would like to thank the Signal Processing Group at the Department of Signals and System at Chalmers for valuable discussions, feedback and supply of the GPU. Finally, I would also like to thank friends and family for proofreading.

Henrik Wallenius, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1
Introduction

The interest of autonomous vehicles has increased substantially during the last few years. Letting a system control a vehicle's interactions with its environment and the vehicle itself has multiple possible benefits, such as increased fuel efficiency, more efficient parking and increased mobility for non-drivers [24]. Furthermore, the number of traffic accidents may be decreased by removing the risk of a human error. A report from the U.S. Department of Transportation claims that the critical reason behind the accident was in 94% of the cases assigned to the driver [35]. Having a system to monitor the driver is therefore of great interest to reduce the number of road traffic victims. Google has taken a step further and plans to by 2020 release Google car, a fully automatic vehicle without steering wheel or pedals. Instead it will be operated by a smartphone [21]. By 2017 Volvo Cars plans to have 100 self-driving cars, used by real-world customers, driven on the ring roads of Gothenburg that will not require human supervision.

1.1 Background

In order to control all tasks of an autonomous vehicle, a combination of several systems is needed. Radars have been tested to detect obstacles and vehicles from far distance [40] while a laser scanner can be used to sweep over the neighborhood of the vehicle. This can be used to discover pedestrians and distinguish between the road and curb when parallel-parking [38]. Another technique is to create a network consisting of the Global Positioning System (GPS) in each car in order to analyze how drivers interact on an infrastructural level in order to predict and prevent traffic congestion. A GPS signal may also determine the position of the vehicle, however due to the lack of accuracy in consumer-grade devices and its lack of ability of acquiring measurements in e.g. tunnels [28], a more reliable system is needed. One solution is to use the video stream from a camera to decide the position of the vehicle. This can be achieved by applying Simultaneous Localization And Mapping (SLAM) which addresses the problem of acquiring a spatial map of the environment while simultaneously localizing the vehicle relative to this model [39].

The map created by SLAM can be built upon several approaches. One approach is through maps containing features. A feature is a local description of an interest point, e.g. a corner, found in an image. By tracking these features in consecutive image frames in a video stream, the 3D-position of these interest points can be estimated given some optimization constraint. The final result is an optimized 3D-map of the discovered features. When returning and driving through the same

location another time the map can be used to position the vehicle. This is done by relating and matching the currently detected features with the features stored in the map. When using this approach, it is essential to construct a reliable map such that it actually reflects the true surrounding. The detected interest points are from now on referred to as feature points and they are detected by a detector.

1.2 Problem Formulation

Current methods of finding and describing features are handmade, rely on a mathematical foundation, e.g. difference of Gaussian and require the user to set some parameters. It is desirable for the features to be invariant with respect to e.g. illumination, scale, rotation and noise such that the feature can be represented uniquely and independently of field of view and time of observation. Otherwise there is a risk of representing the same feature point in multiple ways depending on the external conditions. The traditional feature description algorithms do not cover all mentioned aspects, and creating a map to position via, in a robust manner, is hard due to the ambiguity. The descriptor problem becomes how to construct robust and reliable features, invariant of changes in the environment and view point.

The detected feature point shall not only be invariant but also belong to an object that will always be at the current location. It is therefore also important to classify each feature point in order to know what kind of object it is placed on. When creating a positioning map for an autonomous vehicle it is not advantageous to store features from unstable classes, e.g. a pedestrian or the leaf of a tree; these will probably not be at the same location when returning next time. Instead the features should be based on objects from stable classes, e.g. buildings and road signs. The classification problem becomes how to classify detected feature points in order to determine whether they belong to a stable class and are suitable, or if they are unstable and not suitable.

1.3 Related Work

A requirement for a descriptor is repeatability, the capability of finding and describing the same feature point and its feature description in a unique way despite changes in the environment. Finding descriptors invariant with respect to scale can be treated with a scale-space kernel to search for stable feature points in scale-space. It has been shown by Lindeberg [23] that the only possible scale-space kernel is the Gaussian. Mikolajczyk found that the optima from the Laplacian of Gaussian produced the most stable image features compared to other suitable image functions [26]. These findings were the starting point of the Scale Invariant Feature Descriptor (SIFT) by Lowe [25] where the Laplacian of Gaussian in scale-space is approximated, which reduces the need for computational resources. In SIFT, a histogram of image gradients is constructed and converted to a vector of float numbers which corresponds to the descriptor. By constructing the descriptor via a normalized histogram, SIFT is to some extent also illumination and rotation invariant [25]. Searching in all scale spaces and representing the vectors with float precision

is however computationally expensive and SIFT is hard to utilize in a real-time application.

The complexity of a descriptor can be reduced by introducing a binary approach. Rublee *et al.* [30] compiled the Oriented FAST and Rotated BRIEF (ORB) descriptor, a binary rotation invariant descriptor which is about two orders of magnitude faster than SIFT. The computational cost of comparing two descriptors is also reduced since in the binary case the comparison is on the form of a bitwise XOR operation. Binary descriptors are applicable in state-of-the-art real-time applications, e.g. monocular SLAM solvers [27]. However, using a binary approach confines the representation of a descriptor.

Instead of decreasing the descriptor's complexity, research has been done on how to improve its descriptional power. Zagoruyko *et al.* [43] proposed a deep Convolutional Neural Network (CNN) approach to determine similarities between two patches. Different architectures were tested e.g. the two patches were fed to the same network or in parallel through two identical networks. The final part of the network consisted of a decision network to determine the similarity between the two patches. Simo-Serra *et al.* [33] proposed a method where the decision network as a similarity measurement was removed and replaced by an L_2-norm. They also proposed a training procedure where only the most difficult patches were used when training the network to improve performance. The models generalized well over different datasets, handles illumination changes well and can be used as an in-replacement to SIFT. The networks were however trained on image patches samples on 3D reconstructions and not on real images.

Feature descriptors from e.g. SIFT have been used in classification tasks. Ayers *et al.* [1] used SIFT descriptors for indoor room classification to distinguish between seven room types. Each image was represented by a histogram of the SIFT descriptors in the image and this representation was classified with a Support Vector Machine (SVM) [9] and Adaboost [13]. However, the classifier had problems to make the right predictions as the number of room types increased. State-of-the-art techniques to classify images rely today on convolutional neural networks. Top performing methods on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [31], a competition where over 1 000 000 images from 1 000 classes are classified, are all convolutional neural networks [17, 34, 37]. State-of-the-art techniques to segment and classify pixel by pixel in an image also rely on convolutional neural networks. Since each pixel is classified this technique can be used to classify feature points by mapping the network's prediction to the predefined partition of stable and unstable classes. Top performing approaches on the Cityscapes dataset pixel-level semantic labeling task [8] all propose a solution using convolutional neural networks [22, 42].

1.4 Purpose

The purpose of this thesis is to investigate if a convolutional neural network can be used to learn a feature descriptor to be invariant with respect to scale, rotation and noise in urban street scenes. Furthermore, the purpose is also to explore if a convolutional neural network can be used to classify feature points according to a predefined division.

1.5 Proposed Solution

In this thesis a trained descriptor through a convolutional neural network is proposed as solution to the descriptor problem formulated in Section 1.2. The descriptor is trained using pairs of patches, cropped images, extracted from urban street scenes. Pair of patches from the same feature point form a positive set and patches not from the same feature point form a negative set. To make the descriptor invariant with respect to certain transformations, the patches will be manipulated to handle such cases. The network is trained to distinguish between positive and negative pairs of patches and from the trained network a descriptor is extracted which can be used to describe single patches. Previous work within this field has not dealt with real images, which is now taken into consideration.

Secondly a classifier is trained through a convolutional neural network as solution to the classifying problem formulated in Section 1.2. Patches are extracted and classified as either a stable or unstable patch, using annotated images as reference. A stable patch is from a feature point belonging to an object one assumes will always appear at the site such as a traffic sign or a building and hence suitable to be represented in a positioning map. An unstable patch has a feature point from an object one assumes will not always appear such as a pedestrian or a car. The patches are collected from urban street scenes and the separation between suitable and non-suitable objects are done in a supervised manner before training the network. Using annotated images to train a convolutional neural network classify feature points has not, to my knowledge, been done before.

1.6 Dataset

The dataset used in this thesis is the Cityscapes dataset [8] by Daimler AG R&D, TU Darmstadt, MPI Informatics and TU Dresden. It contains a diverse set of colored stereo video sequences recorded in street scenes and consists of 25 000 frames of size $2\,048 \times 1\,024$, taken from 50 different cities, at different time of the year (spring, summer, fall). The images are annotated such that each pixel is classified to one out of 30 classes, e.g. pedestrian, car, vegetation, road, traffic light and bridge. The annotation is constructed by describing each object through a polygon. 5 000 of the images have a high quality pixel-level annotation while the other 20 000 images have a coarse annotation. The method used in this thesis requires finely annotated images and therefore only the high quality pixel-level annotated images will be used. An example of an image and its annotated counterpart are shown in Fig. 1.1a and 1.1b.

1.7 Scope

Finding and describing features require both a detector and a descriptor. This thesis will only deal with the descriptional part and traditional methods will be used to find feature points. Hence the CNNs will be trained by data extracted from the

(a) **(b)**

Figure 1.1: **(a)**: Example of image from an urban street scene in the Cityscapes dataset. **(b)**: Example of high quality pixel-level annotated image. There are in total 30 classes. Example of classes present in this case are car, pedestrian, building and vegetation.

result of these traditional detector methods. The partition of classes used to train the classifying CNN is done in a supervised manner in advance.

1.8 Structure of the Report

The following parts of this thesis are organized as follows. Chapter 2 presents the theoretical foundation this thesis relies on. It is divided into two main paths, an introduction of concepts in neural networks and a presentation of two popular handmade descriptors. Chapter 3 describes the methods being used. The main parts are developing a descriptor and a classifier and covers how the methods were implemented, what programming language was used and how different parameters were set. The baseline methods used to compare the compiled methods are also explained here. Chapter 4 handles the results. These are discussed in Chapter 5. The thesis is finally concluded in Chapter 6.

2
Theory

This chapter aims to present the theoretical components used in this thesis. The foundation lies in neural networks, which will be presented in its simplest form through a feed forward neural network. The currently used state-of-the-art techniques in neural networks when solving computer vision and image processing problems are described and their training procedure is explained. The second part of the chapter presents two commonly used feature detectors and descriptors.

2.1 Neural Networks

Modelling the behaviors of a complex system, such as face recognition, using traditional mathematical methods is hard. Another procedure to tackle these issues is by mimicking the structures from nature where problems have been solved by evolving a measurement. One example is the human nervous system which receives stimuli from external sources. The stimuli are transduced to electrical impulses and fed to the human brain where it is processed and converted to new electrical impulses into discernible responses as system output through the effectors [16]. The electrical impulses are transmitted from one neuron to the next through synapses, where the electrical signal is converted to a chemical signal and then back to an electrical. When the system is exposed to a certain stimulus caused by its surrounding environment it either strengthens or weakens the activity of the synapse depending on how often the activity occurs. This phenomenon is referred to as syntactic plasticity and enables the human nervous system to adapt to its environment. Connecting thousands and thousands of neurons with synapses enables the human being to handle and solve highly non-linear and complex tasks. This concept of a system evolved with respect to its environment is applicable in many computer-related problems. The mimic of the human brain is often referred to as a neural network. In this section, the foundation of a neural network will be presented as well as the state-of-the-art techniques currently used.

2.1.1 Feed Forward Neural Networks

The fundamental unit in a neural network is a neuron. The concept of a neuron is presented in Fig. 2.1 where the network receives an input $\mathbf{x} = \{x_1, \ldots, x_i, \ldots, x_m\}$ of dimension m. The i:th input node, x_i, connected to the k:th output node, gets weighted with the synapse weight ω_{ki}. Notice that, compared to the synapse in the human brain, the synapse weights can obtain negative values. The weighted input

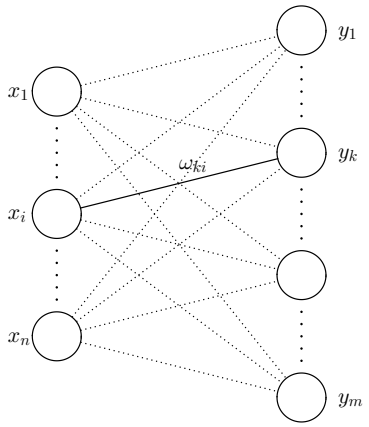


Figure 2.1: Illustration of a simple neural network. Input node x_i and output node y_k are connected through the weight ω_{ki}, denoted by the solid line, and an activation function $\phi(\cdot)$.

is then summed together with a bias term b_k through an adder such that the output u_k from the adder becomes,

$$u_k = b_k + \sum_{i=1}^{m} \omega_{ki} x_i, \tag{2.1}$$

or by adding a fixed input $x_0 = 1$ and synapse weight $\omega_{k0} = b_k$ we obtain

$$u_k = \sum_{i=0}^{m} \omega_{ki} x_i. \tag{2.2}$$

u_k is then transferred to a non-linear activation function $\phi(\cdot)$ in order to enlarge the set of functions the network can map against. Examples of commonly used activation functions [16] $\phi(u_k)$ are the step function mapped to $\{0, 1\}$,

$$\phi(u_k) = \begin{cases} 1 & \text{if } u_k \geq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.3}$$

sigmoid function mapped to $[0, 1]$,

$$\phi(u_k) = \frac{1}{1 + e^{-u_k}}, \tag{2.4}$$

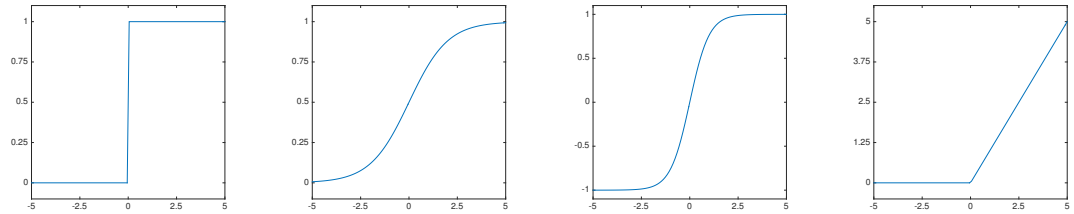
and hyperbolic tangent function mapped to [-1,1],

$$\phi(u_k) = \tanh(u_k) = \frac{e^{u_k} - e^{-u_k}}{e^{u_k} + e^{-u_k}}. \tag{2.5}$$

The activation functions are shown in Fig. 2.2a, 2.2b and 2.2c.

Rectified Linear Units (ReLU) is an activation function which compared to the traditional sigmoid and hyperbolic tangent function maps to an infinite set, $[0, \infty)$, through the function,

$$\phi(x) = \max(0, x) \tag{2.6}$$



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

Figure 2.2: Example of activation functions $\phi(\cdot)$. **(a)**: Threshold function as defined in Eq. (2.3). **(b)**: Sigmoid function as defined in Eq. (2.4) with $a = 1$. **(c)**: Hyperbolic tangent function as defined in Eq. (2.5). **(d)**: ReLU function as defined in Eq. (2.6).

hence constantly zero for $x \in (-\infty, 0)$ and linear for $x \in [0, \infty)$, see Fig. 2.2d.

The output of the neuron after applying the activation function becomes,

$$y_k = \phi\left(u_k\right) = \phi\left(\sum_{i=0}^{m} \omega_{ki} x_i\right), \tag{2.7}$$

hence a weighted linear combination of the input mapped to an interval through a non-linear activation function. The neurons form a layer which can be fed to an additional layer and so on. Data is however not transferred back. This kind of network is referred to as a feed forward neural network (FFNN). When all neurons in the previous layer are connected to all neurons in the current layer the network is referred to a fully connected neural network (FCNN).

2.1.2 Convolutional Neural Networks

A drawback with a FCNN is the growth of number of neurons as the input size increases. Imagine a color image with three color channels and a weight and height both larger than 600 pixels, treating the input image with a similar approach of a neural network where each color pixel corresponds to an input neuron. Assuming the first layer consists of n_1 neurons, the network would require $3 \times 600 \times 600 \times n_1 = 1\,080\,000 n_1$ weights if each input neuron is connected to each neuron in the first layer. The complexity gets unmanageable as n_1 grows. A sparser approach is through a convolutional neural network (CNN) where only local interactions are considered between neurons. It is inspired by the receptive fields of a visual system. Each receptive field is a subarea of the retina from which responses can be elicited by light [2].

A CNN has been proven to tackle computer vision and image processing tasks in an efficient manner. He *et al.* [17] won the 1st place in ILSVRC 2015 object localization task [31] which consists of classifying images among 1 000 categories. In this section the foundations of a convolutional neural network will be explained, considering an image as input. Firstly, two commonly used layers, convolutional layer and max pooling layer, are presented and lastly two loss functions, describing which objective the network shall minimize, are explained.

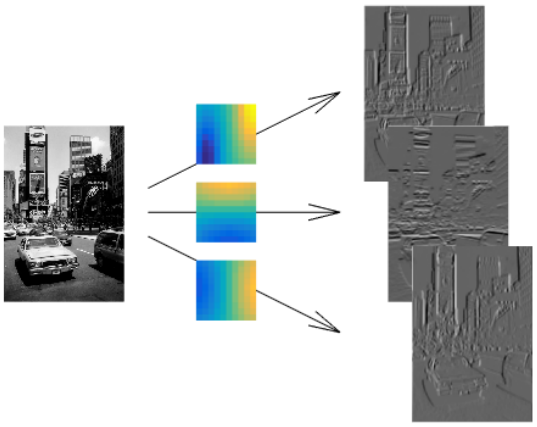


Figure 2.3: Illustration of a convolutional layer. The input is a gray scale image [3] and the layer consists of three kernels, the weights for each kernel are presented with its corresponding heat map. The outputs are stacked on top of each other and fed forward to the next layer of the network. Notice how each kernel finds different features in the image.

2.1.2.1 Convolutional Layer

A convolutional layer used in image processing tasks consists of three dimensions, width n_w, height n_h and depth n_d. Hence the size of the input to the layer is $n_w \times n_h \times n_d$. A kernel K of size $m \times m \times n_d$ with $m^2 n_d$ weights is applied to the input such that a weighted summation occurs among all input neurons inside the kernel including the entire depth. The procedure is equivalent to the weighted summation of the neurons in a fully connected layer as described in Section 2.1.1 but the summation is applied in three dimensions instead of one and only to a window of the input instead of the entire. The output from each summation is a scalar. The kernel is shifted one step in either width or height direction and the procedure is repeated until the kernel has covered the entire image. Moving the kernel over the input refers to the convolutional approach of processing data. The final output is a two dimensional map of size $(n_w - m) \times (n_h - m)$. Several kernels K_i, where $i \in \{1, \ldots n_{out}\}$ can be applied and for each a two dimensional map is obtained. Stacking these n_{out} maps on top of each other the final output from the convolutional layer is of size $(n_w - m) \times (n_h - m) \times n_{out}$. The output is affected by e.g. a non-linear activation function before it is fed to the next layer. In this manner the size of the output can be freely set. Instead of moving the kernel only one step between each convolution the number of steps can be set arbitrarily conditional that the kernel can cover the entire image without moving outside of it. This will result in a less dense map. It is also possible to obtain the same spatial dimensions, $n_w \times n_h$, by adding an $m/2$ wide border of zeroes around the input such that the input becomes $(n_w + m) \times (n_h + m) \times n_d$. An example of three convoluted outputs using three kernels on a gray scale image (the depth of the input is one) is shown in Fig. 2.3. The weights of each kernel is presented through each heat map.

After each convolutional layer a non-linear activation function is applied to each neuron. A commonly used activation function is the ReLU. One reason is the sparser representation it achieves compared to the sigmoid and hyperbolic tangent function. When the network is randomly generated, the probability that the activation function $f(x) > 0$ is 0.5. This implies that only, on average, 50% of the input propagates through the layer. Another advantage of using a ReLU is the avoidance of vanishing gradients which prevents the network from learning. It has also been shown by Krizhevsky *et al.* [20] how ReLUs speeds up the convergence of a convolutional neural network. By replacing the hyperbolic tangent functions with ReLUs, a training error rate of 0.25 was reached six times faster when training on the CIFAR-10 data set [19].

2.1.2.2 Max Pooling

Max pooling is a layer desirable to use in order to reduce the complexity of the network. The input feature map is transformed into several non-overlapping sub-regions. From each sub region is the maximum value extracted and the collection of these forms the output feature map. Imagine a small translation of the original input. A max pooling layer will not be affected since only the maximum value within each sub region is propagated to the next layer. Hence, a max pooling layer is invariant with respect to small translations by reducing the resolution of the feature map [32]. A spatial kernel $K_{m\times m}$ of size $m \times m$ determines the size of the sub regions. Applying a max pooling layer with a kernel of size $m \times m$ on a feature map of size $n \times n$ results in an output of size $\frac{n}{m} \times \frac{n}{m}$ conditional that $\mod(n, m) = 0$.

2.1.2.3 Loss Functions

Depending on what purpose the CNN has, it needs do be trained with respect to a certain criterion. A loss function at the end on the network describes the total error of the networks output with respect to its desired output. Given this error, the loss function tells how the weights should move in weight space to reduce the error and hence improve the performance of the network. For a classification task it is common to add some fully connected layers at the end of the network which final output size is the number of possible classes. The output is mapped such that its sum is unity, hence each output neuron represents the probability of the input belonging to the corresponding neuron's class. The normalization is performed with a *logsoftmax* function defined as,

$$f_{logsoftmax}(y_i) = \frac{e^{y_i - s}}{\sum_j e^{y_j - s}} \tag{2.8}$$

where y_i is the output for the i:th class before normalization, $f_{logsoftmax}(y_i)$ is the probability of the i:th class and $s = \max_j y_j$ is the shift. The loss function $\mathcal{L}_{class}$ to minimize for a classification problem becomes,

$$\mathcal{L}_{class}(\mathbf{y}, c) = -y_c, \tag{2.9}$$

where $\mathbf{y} = [y_1, \ldots, y_n]^T$ is the output for all n classes and $c \in \{1, \ldots, n\}$ is the ground truth class. In order to minimize the loss, it is beneficial for y_c to tend to 1

for $i = c$ and tend to 0 for $i \neq c$.

Instead of classifying, the network can be used to determine whether two inputs are similar or not. A suitable loss function to use is Hinge embedded criterion [33]. The loss function $\mathcal{L}_{sim}$ to minimize is defined as,

$$\mathcal{L}_{sim}(y_i, z_i) = \begin{cases} y_i & \text{if } z_i = 1 \\ \max(0, M - y_i) & \text{if } z_i = -1 \end{cases}, \tag{2.10}$$

where y_i is a measurement between the two inputs, M is a predefined constant, and z_i is the ground truth. z_i is 1 if the two inputs are similar and -1 otherwise. In order for the loss to be minimized it is beneficial for y_i to tend to 0 for similar inputs and to M or larger for non-similar inputs.

2.1.3 Training

The weights and biases of a network given a loss function can be optimized through a training procedure. The compiled dataset the network shall handle is normally divided into three groups, training, validation and testing. The training data, which contains the ground truth, is fed to the network and the network adjusts to it such that the error with respect to the loss function is minimized. The validation data, which contains the ground truth, is fed to the network to examine its performance but does not affect the network's weights. If the network gets too specialized on the training data, it loses its ability to generalize to similar data [16]. The phenomenon is called overfitting and occurs when the network improves on the training data but deteriorates on the validation data. By validating the network through a validation set this can easily be discovered. The test data contains the ground truth and is used for testing and benchmarking the network but is not used during training.

Instead of feeding the entire training dataset to the network the set can be sent in partitions, called *mini-batches*. It is assumed that minimizing the network on one mini-batch is similar to minimizing on the entire training dataset [4]. A technique used to train networks on mini-batches is stochastic gradient descent and is explained in this section. The layers of the network are optimized such that the weights at the output layer is adjusted first and then by moving backwards the layers are adjusted one by one. This procedure is called backpropagation and is presented in more detail in this section.

2.1.3.1 Stochastic Gradient Descent

Given an input x and a desired output y, pairs $\zeta = (x, y)$ are constructed. The loss function for the network is $l(\hat{y}, y)$ where $\hat{y}$ is the prediction from the network and y is the ground truth. Let $\mathcal{F}$ be a family of functions $f_\omega(x)$ parameterized by the weight vector ω describing the weights of the network. Our mission is to find a function $f \in \mathcal{F}$ that describes our problem correctly. By assuming the training data can represent this we want to minimize the loss $Q(\zeta, \omega) = l(f_\omega(x), y)$, where $f_\omega(x)$ is the function mapping the input x to the predicted output $\hat{y}$ [4]. Given the training set consists of n sample pairs of ζ we want to find the optimal ω^* such that empirical risk,

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f_\omega(x_i), y_i), \tag{2.11}$$

is minimized. A commonly used method to optimize problems as (2.11) is gradient descent. The weight vector ω is iteratively updated from iteration t to $t+1$ as,

$$\omega_{t+1} = \omega_t - \gamma\frac{1}{n}\sum_{i=1}^{n} \nabla_\omega Q(\zeta_i, \omega_t), \tag{2.12}$$

where ∇_ω is the gradient operator with respect to ω and γ is a constant learning rate. Calculating the gradients is computationally expensive and instead the gradient can be estimated by randomly selecting a subset Ω of the samples ζ_i such that the update of ω at iteration t to $t+1$ instead becomes,

$$\omega_{t+1} = \omega_t - \gamma\frac{1}{||\Omega||}\sum_{i:\zeta_i\in\Omega} \nabla_\omega Q(\zeta_i, \omega_t), \tag{2.13}$$

where $||\Omega||$ is the cardinality of the set Ω. In its simplest form does Ω contain one sample, ζ_t, and $||\Omega|| = 1$ [4]. The learning procedure can be accelerated through momentum [36]. Instead of only consider ω_t also ν_t is added such that the loss $Q(\zeta, \omega_t)$ is minimized through,

$$\nu_{t+1} = \mu\nu_t - \gamma\frac{1}{||\Omega||}\sum_{i:\zeta_i\in\Omega} \nabla_\omega Q(\zeta_i, \omega_t) \tag{2.14}$$

$$\omega_{t+1} = \omega_t + \nu_{t+1}, \tag{2.15}$$

where γ is the learning rate and $\mu \in [0, 1]$ is the momentum. Momentum can be interpreted as an accelerating gradient descent that accumulates previous gradients as a velocity vector ν_t which decays by a factor μ at each iteration. A further modification of momentum is Nesterov's accelerated gradient (NAG) [36] where the weight update is,

$$\nu_{t+1} = \mu\nu_t - \gamma\frac{1}{||\Omega||}\sum_{i:\zeta_i\in\Omega} \nabla_\omega Q(\zeta_i, \omega_t + \mu\nu_t) \tag{2.16}$$

$$\omega_{t+1} = \omega_t + \nu_{t+1}. \tag{2.17}$$

NAG behaves more stably than the classical momentum in many situations, especially when μ is large [36]. Notice that the original stochastic gradient descent method is obtained when setting $\mu = 0$.

The growth of the network's weights can be limited by a weight decay factor. Weight decay has been important for the CNN to learn [20]. Denoting the loss function as

$$E_0(\omega_t) = \frac{1}{||\Omega||}\sum_{i:\zeta_i\in\Omega} Q(\zeta_i, \omega_t + \mu\nu_t), \tag{2.18}$$

an extra term penalizing large weights is added to the loss function such that the new loss function becomes

$$E(\omega_t) = E_0(\omega_t) + \frac{1}{2}\lambda||\omega_t||^2, \tag{2.19}$$

where λ is the weight decay factor. The weight update using NAG becomes,

$$\nu_{t+1} = \mu\nu_t - \gamma\frac{1}{||\Omega||}\sum_{i:\zeta_i\in\Omega}\nabla_\omega Q(\zeta_i, \omega_t + \mu\nu_t) - \gamma\lambda\omega_t \tag{2.20}$$

$$\omega_{t+1} = \omega_t + \nu_{t+1}. \tag{2.21}$$

2.1.3.2 Backpropagation

Backpropagation is a method of training a network. A sample is fed through the network and the weights are adjusted, starting at the output layer and moving backward, such that the loss is minimized. The procedure differs for different layers but the concept is similar for all. In this section we have focused on the backpropagation of a fully connected layer.

The weighted sum $v_j(n)$ corresponding to neuron j with y_i as input (output from the previous layer) and weights ω_{ji} (connected from neuron i to j) becomes

$$v_j(n) = \sum_{i=0}^{m}\omega_{ji}(n)y_i(n), \tag{2.22}$$

where $n \in \{1, \ldots, N\}$ is the batch set. The output of neuron j becomes

$$y_j(n) = \phi_j(v_j(n)). \tag{2.23}$$

The error at neuron j is defined as $e_j(n) = d_j(n) - y_j(n)$, where $d_j(n)$ is the expected output at neuron j. The error energy at neuron j is defined as $\mathcal{E}_j(n) = \frac{1}{2}e_j^2(n)$ and the total error energy for the layer as

$$\mathcal{E}(n) = \sum_{j\in C}\mathcal{E}_j(n) = \sum_{j\in C}\frac{1}{2}e_j^2(n), \tag{2.24}$$

where C is the set of neurons in the output layer. The goal is to find ω_{ji} such that $\mathcal{E}(n)$ is minimized. Using chain rule we obtain,

$$\frac{\partial\mathcal{E}(n)}{\partial\omega_{ji}(n)} = \frac{\partial\mathcal{E}(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial\omega_{ji}(n)}. \tag{2.25}$$

We obtain $\frac{\partial\mathcal{E}(n)}{\partial e_j(n)} = e_j(n)$, $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$, $\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_j'(v_j(n))$ and $\frac{\partial v_j(n)}{\partial\omega_{ji}(n)} = y_i(n)$. In total we have

$$\frac{\partial\mathcal{E}(n)}{\partial\omega_{ji}(n)} = -e_j(n)\phi_j'(v_j(n))y_i(n). \tag{2.26}$$

The weight ω_{ji} is updated accordingly to stochastic gradient descent,

$$\omega_{ji}^{t+1} = \omega_{ji}^t - \eta\frac{\partial\mathcal{E}(n)}{\partial\omega_{ji}(n)}, \tag{2.27}$$

where η is a positive learning rate. When neuron j belongs to the output layer determine $e_j(n)$ is straight forward since the ground truth $d_j(n)$ is known. When

neuron j does not belong to the output layer the ground truth of the output from that layer is not predefined. To model this let us define the local gradient

$$\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -e_j(n)\phi_j'(v_j(n)), \tag{2.28}$$

which according to chain rule becomes

$$\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \frac{\partial \mathcal{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}\phi_j'(v_j(n)). \tag{2.29}$$

Left to do is determine $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$. Since $\mathcal{E}(n) = \sum_{k\in C}\frac{1}{2}e_k^2(n)$ we have,

$$\begin{aligned}
\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_{k\in C} e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} && (2.30)\\
&= \sum_{k\in C} e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)} && (2.31)\\
&= \left\{ \begin{array}{c} e_k(n) = d_k(n) - \phi_k(v_k(n)) \\ v_k(n) = \sum_{i=0}^{m}\omega_{kj}(n)y(j) \end{array} \right\} && (2.32)\\
&= \sum_{k\in C} \underbrace{(-e_k(n)\phi_k'(v_k(n)))}_{\equiv\delta_k(n)}\omega_{kj}(n). && (2.33)\\
& && (2.34)
\end{aligned}$$

Insertion is Eq. (2.28) yields

$$\delta_j(n) = \phi_j'(v_j(n))\sum_{k\in C}\delta_k(n)\omega_{kj}(n). \tag{2.35}$$

Hence, in order to determine the local gradient at neuron j, $\delta_j(n)$, we must know the local gradients $\delta_k(n)$ for all neuron $k \in C$ which are all neurons ahead of neuron j having a directed connection. In this manner the weights are adjusted, starting with the output layer and moving backwards, hence *backpropagation* [16]. The final update rule becomes,

$$\omega_{ji}^{t+1} = \omega_{ji}^{t} - \eta\frac{\partial \mathcal{E}(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = \omega_{ji}^{t} - \eta\delta_j(n)y_i(n). \tag{2.36}$$

2.1.4 Weight Initialization

Training a network with correctly initialized weights has been shown to be crucial [36]. Glorot and Bengio have proposed a method of how the weight initialization should be performed for a deep neural network [14]. The method was expanded by He *et al.* who achieved a robust initialization that particularly considers networks with ReLUs as activation functions. He *et al.* propose the weights for a layer should be sampled from a Gaussian distribution with zero mean and standard deviation of $\sqrt{\frac{2}{n_l}}$ where n_l is the number of weights in to the layer l.

2.2 Feature Detection and Description

It is of great interest in computer vision and image processing tasks to extract interesting information in images in order to understand the scene. The extracted information can be collected by a detector which traditionally tracks either edges, corners or blobs. Line segments or edges can be detected by Sobel or Canny [7] operator which look for discontinuities in pixel intensities among adjacent pixels. A corner can be seen as an intersection between two lines and can be detected by e.g. the Harris corner detector [15]. The Harris corner detector is however sensitive to scale changes [25] and a more commonly used corner detector is the ORB detector, which is presented in this section. A blob is defined as a compact region lighter or darker than its background surrounded by a smoothly curved edge [10], i.e. it has some attributes that are distinguishable compared to its vicinity. A popular blob descriptor is SIFT and is presented in more detail in this section. The representation of a detected corner or blob is through a float or binary vector called feature descriptor. The feature descriptor is desired to be invariant with respect to e.g. illumination, rotation and affine transformation, which makes it possible to relate transformations between images. The ORB and SIFT descriptor are also presented in this section.

2.2.1 SIFT

Scale Invariant Feature Transform (SIFT) is a detector and descriptor by Lowe [25]. A scale space is introduced to find features that are repeatable during different views of the same object. Lindeberg [23] has shown that the Gaussian function,

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}, \tag{2.37}$$

is the only possible space kernel, where σ is the scale of the kernel (the smaller σ the larger scale). Using this kernel, the scale space function $L(x, y, \sigma)$ becomes,

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.38}$$

where $I(x, y)$ is the intensity of the pixel at coordinate (x, y) in image I. Lowe proposed that convoluting the difference of Gaussian

$$D(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma) \tag{2.39}$$

with the image $I(x, y)$ and a factor k in scale difference is an effective method of finding stable scale space extrema. This is because it is a close approximation to the Laplacian of Gaussian $\sigma^2\nabla^2 G$ which produces the most stable image features compared to other possible image functions [25]. Using the distributive property of convolution, we can rewrite $D(x, y, \sigma) * I(x, y)$ as,

$$\begin{aligned} D(x, y, \sigma) * I(x, y) &= [G(x, y, k\sigma) - G(x, y, \sigma)] * I(x, y) && (2.40)\\ &= G(x, y, k\sigma) * I(x, y) - G(x, y, \sigma) * I(x, y) && (2.41)\\ &= L(x, y, k\sigma) - L(x, y, \sigma), && (2.42) \end{aligned}$$

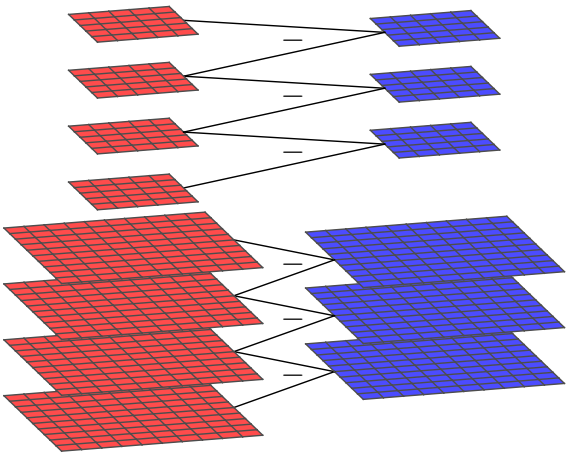


Figure 2.4: Subtraction of adjacent scale spaces, $L(x, y, k\sigma)$ and $L(x, y, \sigma)$, represented as red planes. The difference corresponds to the difference of Gaussian, represented as blue planes. When all neighboring scale spaces for one octave is completed the procedure is repeated for the next one where the size of the scale spaces is reduced by a factor of four.

where k is the factor in scale between two adjacent scales. The scale spaces are divided into octaves such that σ doubles for each, i.e. $k = 2^{1/s}$ if there are s scales per octave. When going to a higher octave only every second pixel in each row and column is taken, reducing the size of the image by a factor of four. The stable scale space extrema are found by determining $L(x, y, \sigma)$ for all examined spaces and subtracting neighboring scale space images to form the set of possible candidates, see Fig. 2.4. Each candidate has nine neighbors in the scale above and below and eight on the same scale. The candidate is considered an extreme if it is larger or smaller than all of its 26 neighbors.

The location of the extreme is however only an estimate since the Laplacian of Gaussian was approximated with the difference of Gaussian. It has been shown that fitting a 3D quadratic function around the candidate and interpolating to find a better estimate of the extreme, increased stability heavily [25]. Making a Taylor expansion of $D(x, y, \sigma)$ translated such that its origin is at the candidate point and denoting (x, y, σ) as $\mathbf{x}$ we obtain,

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}, \tag{2.43}$$

where $\mathbf{x} = (x, y, \sigma)$. The better estimate of the extreme, $\hat{\mathbf{x}}$, is obtained by differentiating Eq. (2.43) and solving for when it is zero (occurs for $\hat{\mathbf{x}}$),

$$\frac{\partial D}{\partial \mathbf{x}} + \frac{\partial^2 D^T}{\partial \mathbf{x}^2}\hat{\mathbf{x}} = 0, \tag{2.44}$$

hence,

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}}^{-1} \frac{\partial D}{\partial \mathbf{x}}. \tag{2.45}$$

Inserting Eq. (2.45) in Eq. (2.43) we obtain

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}}. \tag{2.46}$$

Extrema with a small value of $D(\hat{\mathbf{x}})$ are considered unstable and all extrema below a predefined threshold τ, $D(\hat{\mathbf{x}}) < \tau$, are removed. The final examination of the candidate points is to exclude all detected points on edges. A point on an edge has a large principal curvature across the edge and a small one in the perpendicular direction [25]. The principal curvatures are determined by the ratio of the eigenvalues of the Hessian matrix H,

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}. \tag{2.47}$$

Since we are only interested in the ratio between the two eigenvalues it is sufficient to examine the ratio between them through the trace and determinant of H. Letting λ_α be the largest eigenvalue and λ_β be the smallest the trace and determinant of H are computed by,

$$\mathrm{Tr}(H) = D_{xx} + D_{yy} = \lambda_\alpha + \lambda_\beta \tag{2.48}$$

$$\mathrm{Det}(H) = D_{xx} D_{yy} - D_{xy}^2 = \lambda_\alpha \lambda_\beta. \tag{2.49}$$

Letting r be the ratio between λ_α and λ_β we can express the ratio between $Tr(H)^2$ and $Det(H)$ as,

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} = \frac{(\lambda_\alpha + \lambda_\beta)^2}{\lambda_\alpha \lambda_\beta} = \frac{(r\lambda_\beta + \lambda_\beta)^2}{r\lambda_\beta^2} = \frac{(r+1)^2}{r}. \tag{2.50}$$

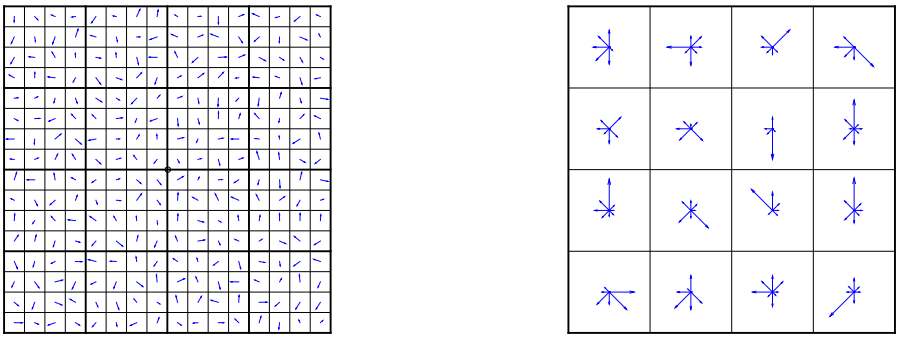
Eq. (2.50) only depends on the ratio r between the two eigenvalues and obtains its minimum for $r = 1$, hence when the eigenvalues are equal. It is therefore sufficient to examine if $\frac{(r+1)^2}{r} < \tau$ to exclude the candidate point. The threshold τ is a tunable parameter and Lowe decided, after some experiments, to set $\tau = 10$ [25].

Each candidate fulfilling all the requirements then gets an orientation representation. Lowe tried different approaches and the one resulting in the most stable representation was through determine the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ for the scale space image $L(x, y, \sigma)$,

$$m(x,y) = \sqrt{(L(x+1,y,\sigma) - L(x-1,y,\sigma))^2 + (L(x,y+1,\sigma) - L(x,y-1,\sigma))^2} \tag{2.51}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1,\sigma) - L(x,y-1,\sigma)}{L(x+1,y,\sigma) - L(x-1,y,\sigma)}\right). \tag{2.52}$$

The gradient orientations for points in a predefined neighborhood of the extreme are calculated. The orientations $\theta(x, y) \in [0, 360)$ degrees form a histogram with 36 bins



(a) **(b)**

Figure 2.5: Concept of descriptor representation by SIFT. **(a)**: Determine gradients of $L(x, y, \sigma)$ for points (x, y) around the candidate (marked as a black circle in the middle). In total there are $16 \times 16 = 256$ neighboring points, divided into 16 regions, which borders are visualized with bold lines. **(b)**: For each region, an orientation histogram with 8 bins is obtained. The histogram is weighted with the magnitude of the gradient and a Gaussian window centered at the feature point. These 16 histograms with 8 bins each form the descriptor of dimension 128.

(10 degrees in each) which is weighted with the gradient's magnitude and a Gaussian window centered at the candidate point. The Gaussian window favors points near the candidate. The largest bin in the histogram corresponds to the orientation of the candidate point and enables SIFT to be rotation invariant since all further calculations can be referred to the orientation. If there is no clear main orientation from the histogram the candidate is discarded. All approved candidate points are the detected feature points by the SIFT descriptor and each is characterized by its location (x, y), scale σ and orientation $\theta(x, y)$.

When the detector has found the feature points each point and its vicinity is represented through a descriptor. A 16×16 grid, centered at the feature point in the scale space image $L(x, y, \sigma)$, is applied. The magnitude and gradient orientation for each box in the grid are determined as in Eq. (2.51) and (2.52), see Fig. 2.5a. The grid is divided into 16 subregions with 16 pixels in each and each region is described with an orientation histogram of 8 bins. The histogram is weighted with the magnitude of the gradient and a Gaussian filter centered at the feature point, see Fig. 2.5b. Combining all 16 subregions and its histogram of dimension 8, a vector of dimension 128 is obtained. By normalizing the vector, the SIFT descriptor becomes invariant to affine illumination transformations [25].

2.2.2 ORB

The detector in ORB is based on the FAST detector [29] which is a corner descriptor where a circle consisting of 16 pixels is applied around each corner candidate point p. p is defined as a corner if at least 12 contiguous pixels at the circle have a pixel intensity larger than the intensity of the candidate point I_p plus a threshold t. The

same holds if the pixel intensity is less than $I_p - t$. Since all 12 contiguous pixels (at least) must fulfill this requirement can a non-corner be decided by only examine 3 pixels [29]. An example of a detection is shown in Fig. 2.6. The FAST detector has no corner measurement. Since edges often also trigger the detector, the ORB detector has a Harris corner measurement. Determine the partial derivatives I_x and I_y for image I and applying a Gaussian window w centered at the examined point (x, y) a Harris matrix, E,

$$E(x,y) = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{bmatrix}, \tag{2.53}$$

is obtained [15]. The Harris corner measurement R is defined as,

$$R(x,y) = \mathrm{Det}(E(x,y)) - k\mathrm{Tr}(E(x,y))^2 \equiv \lambda_\alpha \lambda_\beta - k(\lambda_\alpha - \lambda_\beta)^2 \tag{2.54}$$

where Det denotes determinant and Tr trace, λ_α and λ_β are the two eigenvalues of E and k is a tunable parameter [15]. A corner has a large value of R since a corner corresponds to E having two large eigenvalues while an edge has one large. By setting a low threshold r initially such that only points $R(x, y) > r$ are excepted and gradually increase it until only the N best corner points are found.

Compared to the FAST detector, ORB is rotation invariant. By determining the intensity centroid C of image I,

$$C = \left(\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}} \right) \tag{2.55}$$

where,

$$m_{pq} = \sum_{x,y} x^p y^p I(x,y), \tag{2.56}$$

the orientation is defined as the vector from the corner to the centroid.

The descriptor in ORB relies on BRIEF (Binary Robust Independent Elementary Features) [6]. It is a fast and memory efficient descriptor due to its binary representation. A smoothed patch $\mathbf{p}$ of size $S \times S$ with its center at the detected point is extracted and the test τ is applied such that

$$\tau(\mathbf{p}, \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases}. \tag{2.57}$$

$\mathbf{p}(\mathbf{x})$ is the pixel intensity of the smoothed patch at pixel $\mathbf{x}$. The pixel pairs $(\mathbf{x_i}, \mathbf{y_i})$ are predefined from a Gaussian distribution with 0 mean and a standard deviation of $\frac{1}{25}S^2$ (determined after testing different distributions). The binary descriptor of the detected corner point is defined by,

$$f_n(\mathbf{p}) = \sum_{i=1}^{n} 2^{i-1} \tau(\mathbf{p}, \mathbf{x_i}, \mathbf{y_i}). \tag{2.58}$$

The similarity between two descriptors is easily determined by the Hamming distance, a bitwise XOR operation between two descriptors followed by a bit count.

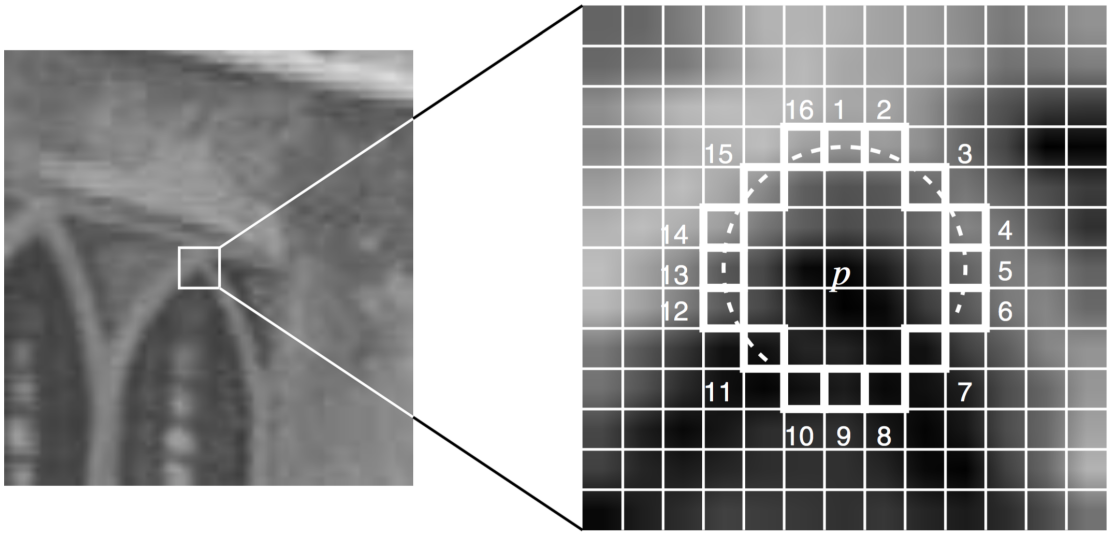


Figure 2.6: Corner detected by the ORB detector since at least 12 contiguous pixels in the circle have a pixel intensity larger than the intensity of the candidate point p plus a threshold t. These pixels are represented with the dotted arc [30].

3
Methods

This chapter presents the methods and implementations compiled to train and evaluate the feature point descriptor and classifier. Firstly, the techniques of how the datasets for training and validation were obtained are explained. Secondly, the models, what the training procedure looked like and how the models were implemented are presented. Lastly the baseline methods to evaluate the models are explained in detail.

3.1 Programming Tools

The data processing to create the training and validation sets were created in MATLAB using its image processing toolbox which has a wide variety of methods to manipulate images. It also has a setup of handmade feature detectors and descriptors. Neither the SIFT detector nor descriptor are implemented in the toolbox however. Instead `VLFeat`, an open source library [41] which contains algorithms to detect SIFT points and describe them is used.

The methods for training the networks in this thesis were developed in `Torch`, a scientific computing framework with support for CNNs based on the programming language `LuaJIT`. There are plenty of deep learning frameworks to choose from and `Torch` was used in this project due to its clearly documented tutorials and large community wiki. Using `Torch` also enabled the training of networks on GPUs provided by the department. The pretrained networks that were used were either developed in `Torch` or converted from another framework. The adjustment of weights and biases through backpropagation was done with the numeric optimization package for `Torch`, `optim`, which enabled stochastic gradient descent with learning rate, momentum, Nestrov's momentum and weight decay among others. Training and validation were performed on the GPU Nvidia Titan X.

3.2 Descriptor

The descriptor is a CNN. Given a cropped area from an image, called patch, the output of the network is a one dimensional vector, hence a feature descriptor. The descriptor is trained with the constraint that similar patches should have similar feature descriptors.

(a) **(b)**

Figure 3.1: Example of patches from the generated dataset. **(a)**: Pair of patches extracted from the same SIFT point creating the positive set. **(b)**: Pair of patches extracted from different SIFT points creating the negative set.

3.2.1 Generating Data

Patches to train and evaluate the learned descriptor are extracted from Cityscapes dataset. Each image in the dataset, after converted to grayscale, is divided into cells by applying a grid of size 12×6. From each cell, the most stable feature point is collected. A SIFT detector is used to find the feature points and the stability is measured by the threshold of the peaks in difference of Gaussian space. By incrementally increasing the threshold, the number of detected feature points will decrease. The procedure is repeated until there is only one feature point left in the cell. In some regions the descriptor will not find any points at the initial threshold. These cells are discarded. The localized feature points will from now on be referred to as SIFT points.

Around each SIFT point a patch of size 64×64 pixels is cropped. For each SIFT point the original grayscale image is duplicated and transformed. The transformed image is used to create similar and non-similar patches to the patch from each SIFT point in the original image. Firstly, the duplicated image is rescaled by a factor $f_s \in \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right]$ (corresponding to a factor 0.5 to 2 in number of pixels), secondly rotated by $f_r \in [-5, 5]$ degrees and lastly added with Gaussian noise with 0 mean and standard deviation of $f_n \in [0.1, 0.5]$ pixel intensity (the pixel intensity in the image is between 0 and 255). f_s, f_r and f_n are all drawn from a uniform distribution in their respective interval. Since the scale and rotation are known the corresponding patch in the duplicated image can easily be determined. The procedure is repeated one additional time for the same SIFT point such that for each point there are three patches. These triplets correspond to the positive set and a combination of one of the patches with any patch not belonging to the triplet construct the negative set. Some examples of positive and negative pairs of patches are shown in Fig. 3.1. With this generated dataset of patches, the objective is to learn the descriptor to be invariant to scale, rotation and noise.

3.2.2 Pretrained Model

Using a CNN to compare image patches has been shown to be successful [33, 43]. Instead of learning a descriptor from scratch a pretrained one can be used. The

descriptor used in this thesis was pretrained on Multi-view Stereo Correspondence (MSC) dataset [5] by Simo-Serra *et al.* [33] and is publicly available. MSC consists of corresponding grayscale 64×64-patches sampled from 3D reconstructions of the Statue of Liberty in New York, Notre Dame in Paris and Half Dome in Yosemite. From the reconstruction, corresponding feature points were found and the patches were extracted from each feature point. The learned descriptor is a CNN with three convolutional layers. Each layer has the hyperbolic tangent as activation function and kernel sizes 7×7, 6×6 and 5×5 respectively. After each convolutional layer an average pooling layer with kernel size 2×2 was applied. The input of the network is a 64×64 normalized grayscale image with mean 0 and standard deviation 1 and the output is a descriptor of size 128.

3.2.3 Training Model

The objective is to distinguish between SIFT points and represent them uniquely. This is equivalent to distinguishing between patches that may or may not be from the same SIFT point. Patches from the same SIFT point should be represented as similar while patches from different ones should not. A technique of dealing with this kind of similarity and dissimilarity problem is through a Siamese network [33].

A Siamese network consists of two identical CNNs connected in parallel, see Fig. 3.2. The data fed to the Siamese network is grouped in pairs such that one of the networks processes one part of the data and the other network the other part. When the CNN is a descriptor the input data consists of pairs of patches as in Fig. 3.1. Since the networks are identical, having the same setup of weights and biases, data is processed in the same way by the two networks. Siamese networks have been used to train feature extraction from patches [33, 43], such that similar patches shall have a similar feature description. The output from each CNN is a one dimensional vector, hence a descriptor. Since similar patches should have similar descriptors it is suitable to use Hinge embedded criterion presented in Section 2.1.2.3,

$$\mathcal{L}_{sim}(p_1, p_2, z) = \begin{cases} ||D(p_1) - D(p_2)||_2 & \text{if } z = 1 \\ \max(0, M - ||D(p_1) - D(p_2)||_2) & \text{if } z = -1 \end{cases}, \tag{3.1}$$

where the similarity measurement is the L_2-norm of the difference between the two descriptors $D(p_1)$ and $D(p_2)$ from patches p_1 and p_2, see Fig. 3.2. Notice that $z = 1$ if the two patches shall be similar and $z = -1$ if not. Notice also that since the two CNNs have the same weight and biases they will be affected exactly the same as backpropagation is executed through the network. When training is completed one of the CNNs can be extracted from the Siamese network and be used analogously to SIFT.

3.2.3.1 Hard Mining

One successful training approach for a Siamese network is hard mining [33]. Instead of backpropagating all patches in the training batch, only the hardest, the pairs with the largest loss term, will affect the weights and biases in the Siamese network. If backpropagation is applied to all negative pairs the network will, once it has reached

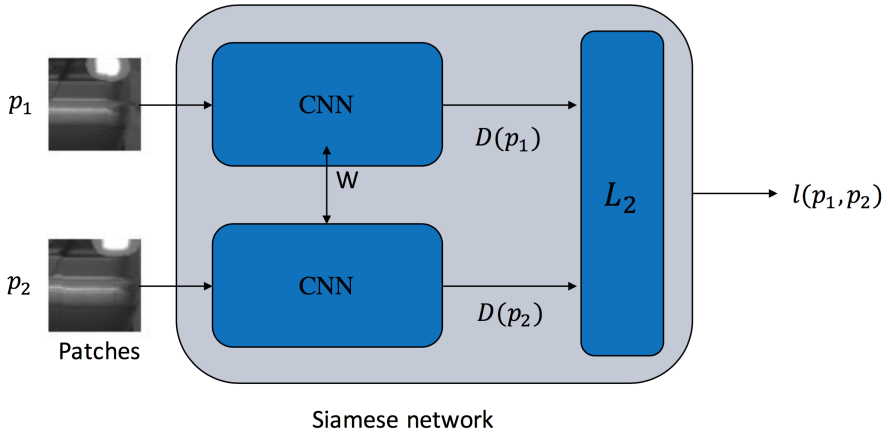


Figure 3.2: Illustration of a Siamese network. Two identical convolutional neural networks (CNN) sharing the same setup of weights and biases W are fed with two patches p_1 and p_2. The output of each CNN is a descriptor, in this case $D(p_1)$ and $D(p_2)$. An L_2-norm together with the Hinge embedded criterion as loss function are used as a similarity measurement between two patches. After training, one of the CNNs can be extracted and used analogously to SIFT.

some level of performance, have an L_2-norm larger than M and hence not contribute to the training [33]. By using the hard mining approach one diminishes the scenario of slowing down the learning process.

3.2.3.2 Receiver Operating Characteristic

Receiver operating characteristic (ROC) curves are popular to use when presenting results from binary decision problems [11]. The binary decision problem encountered for the descriptor is to decide the threshold of the norm between two descriptors being positive or negative. A ROC curve shows the relationship between the number of correctly classified positive pairs with respect to the number of incorrectly classified negative pairs. By sweeping the threshold over an interval the relationship is obtained. The area under the ROC curve, AUROC, is often used as a metric of the performance, the higher the better [5, 11].

3.2.3.3 Implementation

The Siamese network was compiled in `Torch` using a parallel table. This enabled the addition of two CNNs in parallel with shared weights, biases and gradients. The two networks were merged with an L_2-norm and Hinge embedded loss as loss function. These features are implemented in `Torch`. The training and validation data were stored in a four dimensional tensor with dimensions patch number, color channel, width and height. The descriptor was trained with the hard mining approach and without. The hard mining approach consisted of evaluating the loss of 1 024 randomly chosen positive pairs and 1 024 randomly chosen negative pairs. The 128 positive pairs with the largest loss and similarly for the negative set were

backpropagated through the network. When hard mining was not applied all 2048 pairs were backpropagated.

3.2.4 Baseline Method

The baseline method used to compare the learned descriptor was the SIFT descriptor. The procedure of generating the dataset for the baseline method is quite similar to the dataset used for training and validation of the trained descriptor. Given an image from the Cityscapes validation set a grid of size 12×6 was applied and not more than one SIFT point was found in each element by incrementing the peak threshold. Each SIFT point in the original image $\mathbf{p}^1$ was represented with the attributes pixel-position x^1, y^1, scale s^1 and orientation θ^1, hence $\mathbf{p}^1 = \{x^1, y^1, s^1, \theta^1\}$. The original image was then duplicated and transformed by scaling of a factor $f_s \in \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right]$ and rotation by $f_r \in [-5, 5]$ degrees. f_s and f_r were drawn from a uniform distribution.

A SIFT detector was applied on the transformed image with a small peak threshold to obtain many possible matching candidate points to the original image. The search was performed in a brute-force manner where each SIFT point in the original image was compared to each SIFT point in the transformed image. Since the rotation and scale was known, the transformation from $\mathbf{p}^1$ to $\hat{\mathbf{p}}^1$ in the transformed image was easily determined. A match between SIFT point $\hat{\mathbf{p}}^1$ and $\mathbf{p}^2$ was considered when the pixel distance $|\hat{\mathbf{p}}^1_{x,y} - \mathbf{p}^2_{x,y}|_2 < 0.2$, scale difference, $|\hat{\mathbf{p}}^1_s - \hat{\mathbf{p}}^2_s|_1 < 0.1$ scale units and rotation difference $|\hat{\mathbf{p}}^1_\theta - \hat{\mathbf{p}}^2_\theta|_1 < 0.2$ degrees. Using these thresholds, we never encountered multiple SIFT points matching the same SIFT point in the original image.

When all SIFT point pairs had been localized a Gaussian noise, with mean 0 and standard deviation of $f_n \in [0.1, 0.5]$ pixel intensity drawn from a uniform distribution, was applied to the both images. Patches and SIFT descriptors were extracted at the localized SIFT points.

3.3 Classifier

The classifier consists of a CNN and a FCNN. Given a patch p the CNN transforms it to a one dimensional deep feature vector and the FCNN, with the same number of output neurons as classes, classifies the deep feature vector. The patch can either be classified as stable or unstable, their meanings are described in more details in Section 3.3.1. An illustration of the concept of the network is shown in Fig. 3.3.

3.3.1 Generating Data

Feature points were detected in the training and validation set of Cityscapes dataset using the FAST detector. To find points evenly spread over the image, which is desirable in motion estimation tasks [12], the image was partitioned into a grid, of size 14×7, where the detector was applied to each cell. The detected feature points in a cell were sorted with respect to its Harris corner measurement, see Eq. (2.54), and only the feature point with the largest measurement was kept. If the detected

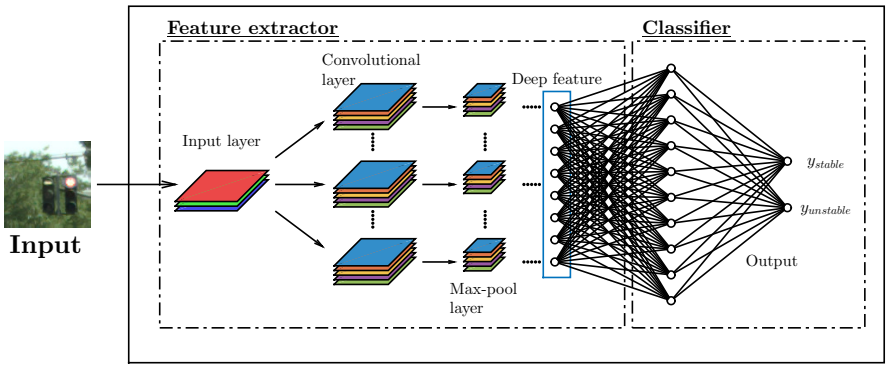


Figure 3.3: Illustration of classification network. A color patch is fed to a convolutional neural network (CNN), hence input to the network has three color channels. The patch is fed to a CNN, containing convolutional layers and max pooling layers, extracting a deep feature vector. A FCNN maps this deep feature vector and its output is the probability of the patch belonging to class stable or unstable.

point and a patch centered around it of size 100×100 were not outside of the image the patch was stored.

In advance the classes according to Table 3.1 were determined to be stable, i.e. suitable for the positioning map, or unstable, not suitable for the positioning map. The partition of classes was motivated by how likely it was for the class to always appear at the scene. E.g. a building or a traffic sign was more likely to appear than a car or a pedestrian. Areas where it was hard to find good feature points, such as asphalt, were also set to be unstable. Hence the class road was treated as unstable despite the well defined lane marks.

The patches for all accepted points were extracted and their labels were determined from the high quality pixel-level annotated images in the Cityscapes dataset. The procedure was performed on 2 975 training images and 500 validation images. Since the test images in Cityscapes do not have accessible annotations, half of the validation images were used as test images. The total set of training patches consisted of 147 939 patches, the validation set of 13 199 patches and the test set of 12 498 patches.

3.3.2 Pretrained Model

Instead of implementing and training a CNN from scratch a pretrained CNN was used. The pretrained network used was VGG Net-E, a 19 layers deep CNN for large-scale image recognition which is publicly available [34]. It was motivated to be a suitable starting point since the network conquered first and second places in the localization and classification tracks respectively in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 [31]. It has been shown that deep image representations from networks trained on ILSVRC generalize well to other datasets, where they have outperformed hand-crafted representations by a large

Table 3.1: Division of annotated classes in Cityscapes dataset. The stable classes are assumed to be more likely to find when returning to the same location another time. The classifier is trained with respect to this partition.

	Classes																								
	Road	Side walk	Parking	Rail track	Building	Wall	Fence	Guard rail	Bridge	Tunnel	Pole	Pole group	Traffic light	Traffic sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Caravan	Trailer	Train	Motorcycle	Bicycle	License plate
Stable					✓	✓	✓	✓	✓		✓	✓	✓	✓														
Unstable	✓	✓	✓	✓						✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

margin [34].

The architecture of the network is presented in Table 3.2. Compared to earlier top performing networks on ILSVRC [20, 44], VGG Net-E has a smaller kernel, 3×3 compared to 7×7 and 11×11 and a smaller stride of 1 compared to 2 and 4. In total there are 16 convolutional layers with kernel size 3×3, 5 max pooling layers of size 2×2 with stride 2, 3 fully connected layers and a soft-max layer. Between each layer is a ReLU activation function injected. Stacking convolutional layers with a small kernel and a non-linear activation function in between can be seen as a decomposition of one convolutional layer with larger kernel. Adding two 3×3 convolutional layers successively is equivalent to one 5×5 convolutional layer and three 3×3 convolutional layers corresponds to one 7×7 convolutional layer. The advantage is the number of parameters required. Three 3×3 convolutional layers with C channels require $3(3^2C^2) = 27C^2$ weights while one 7×7 convolutional layer with C channels needs $7^2C^2 = 49C^2$, hence a reduction by 44.9%. VGG Net-E has either two or four consecutively convolutional layers and between 64 and 512 channels per layer, see Table 3.2 for further information.

The FCNN at the final layers of VGG Net-E was originally trained on the 1 000 classes in ILSVRC. To adapt the network to our purpose the final three fully connected layers were removed and replaced with three fully connected layers according to Table 3.3. From the convolutional part of the classifying network a deep feature vector of size 25 088 was obtained and this was classified on the FCNN.

3.3.3 Training

The objective is to separate patches from stable and unstable classes. Since this becomes a classification problem is the classifier trained with a class loss function as defined in Section 2.1.2.3

$$\mathcal{L}_{class}(\mathbf{y}, c) = -y_c. \tag{3.2}$$

Given a patch p to the network the output, after applying a logsoftmax function, see Eq. (2.8), becomes $\mathbf{y} = [y_{stable}, y_{unstable}]^T$ where y_{stable} is the probability of the patch belonging to the class stable and $y_{unstable}$ of belonging to the class unstable.

Table 3.2: Network configuration of pretrained network VGG Net-E [34]. The input is a 224×244 RGB image and output a $1 \times 1\,000$ probabilistic prediction of the input's class. Conv$x - y$ is a convolutional layer with y kernels of size x. FC-z is a fully connected layer with z neurons. Logsoftmax is a logsoftmax layer to represent the output as probabilistic.

CNN Configuration [VGG Net-E]
Input (224×224 RBG image)
Conv3-64
Conv3-64
Max pool
Conv3-128
Conv3-128
Max pool
Conv3-256
Conv3-256
Conv3-256
Conv3-256
Max pool
Conv3-512
Conv3-512
Conv3-512
Conv3-512
Max pool
Conv3-512
Conv3-512
Conv3-512
Conv3-512
Max pool
FC-4096
FC-4096
FC-1000
Logsoftmax

Table 3.3: Network configuration of fully connected layer after the pretrained convolutional part of VGG Net-E [34]. ReLU has been used as activation function after each fully connected layer and a Logsoftmax is used to convert the prediction to a probability.

Layer	Number of neurons in previous layer	Number of neurons in layer
FC-1	25 088	4 096
FC-2	4 096	500
FC-3	500	40
FC-4	40	2
Logsoftmax	–	–

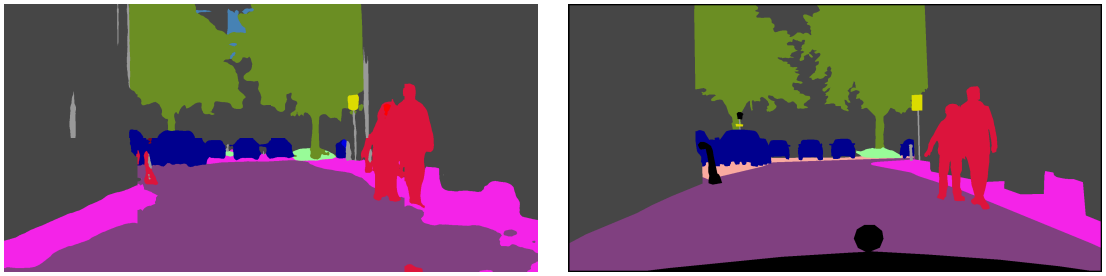
The ground truth is represented by $c \in \{stable,\ unstable\}$, hence y_c is y_{stable} when the training patch is stable and $y_{unstable}$ if unstable.

3.3.3.1 Implementation

VGG Net-E was developed in the deep learning framework `Caffe` and was converted to `Torch` with the package `loadcaffe`. Training and validation data were represented with a four dimensional tensor with dimensions patch number, color channel, width and height and were preprocessed by mean subtraction and standard deviation division (based on the mean and standard deviation over the entire training set). Since the pretrained CNN was not adjusted during training, all patches were preprocessed through the CNN and hence the FCNN was trained on the deep feature vectors to reduce training time. The FCNN was trained with stochastic gradient descent in minibatches of size 256 randomly chosen without replacement (until the entire training set was covered). The weights of the network were initialized using the approach proposed by Glorot and Bengio [14]. After some testing and inspiration from [34] and [20] the following parameters were used; learning rate γ of 0.001, momentum μ of 0.9 (using Nesterov's approach) and weight decay of 0.0005. After each epoch, which is when all minibatches covering the entire training set is tested, the validation set was executed. The training lasted until the error of the validation set had stabilized.

3.3.4 Baseline Method

The baseline method used to compare the classifier was the best performing publicly available model for semantic segmentation on the Cityscapes dataset pixel-level semantic labeling task [8], Dilation10 [42]. Semantic segmentation corresponds to the problem of given an image, label each pixel to one of predefined classes. Dilation10 is a convolutional network module that is specifically designed for dense prediction and is specifically learned on training images from Cityscapes dataset. Since Dilation10 estimates each pixel to one of the classes in Cityscapes it can also classify feature points to one of the two classes our classifier is trained on. The evaluation is performed on the validation set of Cityscapes dataset. In total 500 images were segmented, see Fig. 3.4 for an example and its ground truth. For each segmented image the annotation made at the feature points found when creating the validation patches for the classifier were compared with the ground truth. If Dilation10 and the ground truth classify a feature point to be in the same class (either stable or unstable) the point is annotated correctly and falsely otherwise.



(a) **(b)**

Figure 3.4: **(a)**: Example of segmentation of validation image, on Cityscapes Dataset, produced by Dilation10 **(b)**: Ground truth segmentation of example image.

4
Results

4.1 Descriptor

The Siamese network was trained for 500 iterations using hard mining, where each iteration consisted of testing two batches of size 1 024. An identical network was trained for 500 iterations but without hard mining. After 500 iterations the average error for the validation set for both networks had stabilized. The networks were tested, on a test set containing positive and negative pairs of patches. Histograms of the L_2-distance between the descriptors of the two patches in each pair were created. The histogram when applying hard mining is shown in Fig. 4.1 and without hard mining in Fig. 4.2. A comparison is made using the SIFT descriptor and its corresponding histogram is shown in Fig. 4.3. A ROC graph of the three examined descriptors are shown in Fig. 4.4 and the AUROC measurement is shown in Table 4.1

Table 4.1: AUROC measurement of the three tested descriptors.

Descriptor	AUROC
Deep descriptor with hard mining	0.990
Deep descriptor without hard mining	0.966
SIFT descriptor	0.962

4.1.1 Runtimes

The runtimes of the learned deep descriptor and the SIFT descriptor were evaluated by measuring the execution time of calculating the feature descriptors given the position of the feature point and its scale and orientation. The mean runtime of calculating one descriptor is shown in Table 4.2

Table 4.2: Average runtime of calculating one descriptor given the position of the feature point and its scale and orientation in milliseconds.

Descriptor	Average runtime [ms]
Deep descriptor (GPU)	0.858
Deep descriptor (CPU)	36.958
SIFT descriptor (CPU)	0.258

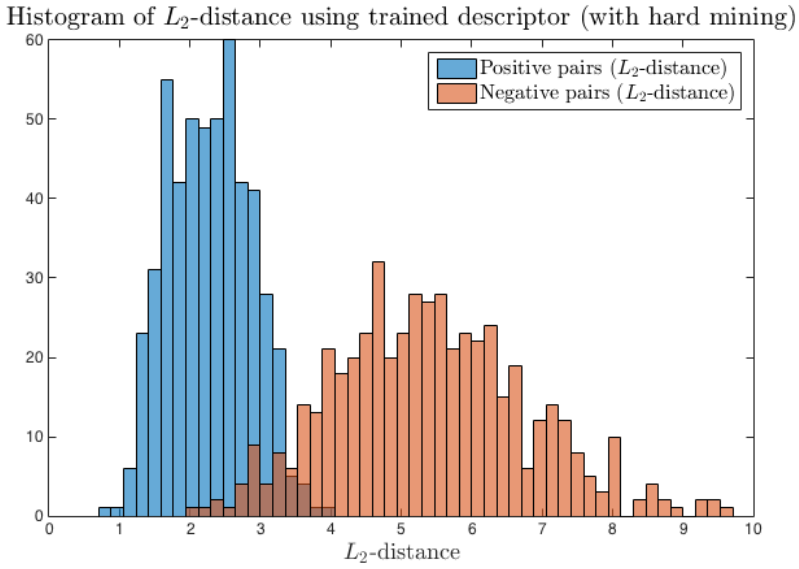


Figure 4.1: Histogram of L_2-distance between the descriptors from each pair using the trained descriptor with hard mining, evaluated on the test set. The blue bars show the distribution of the L_2-distance for the positive pairs and the orange colored bars for the negative pairs. The brown colored area shows the overlap between the two histograms.

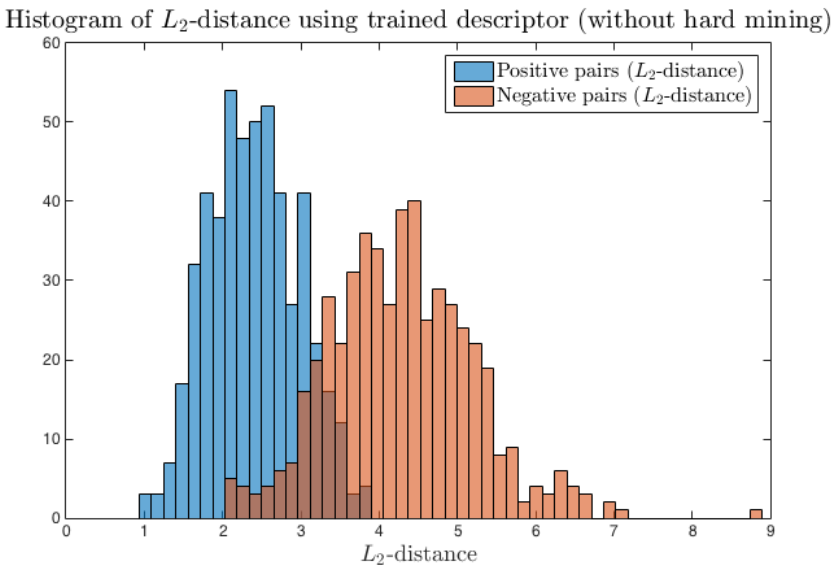


Figure 4.2: Histogram of L_2-distance between the descriptors from each pair using the trained descriptor without hard mining, evaluated on the test set. The blue bars show the distribution of the L_2-distance for the positive pairs and the orange colored bars for the negative pairs. The brown colored area shows the overlap between the two histograms.

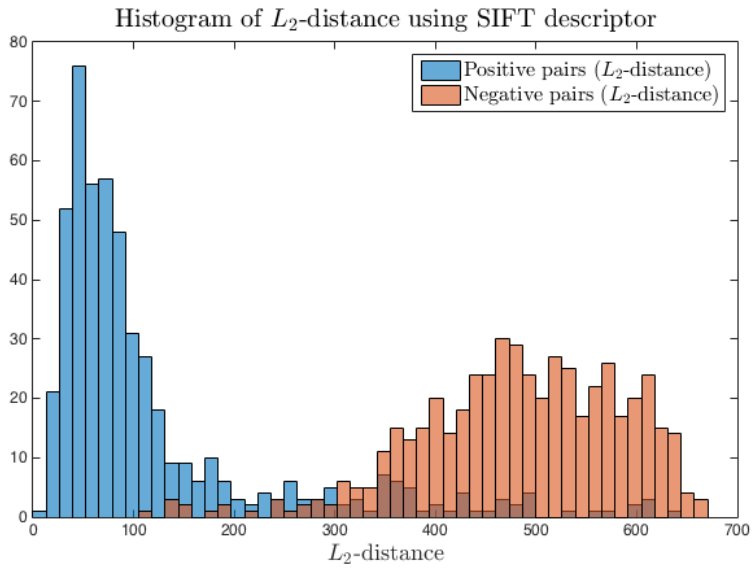


Figure 4.3: Histogram of L_2-distance between the descriptors from each pair using the SIFT descriptor, evaluated on the test set. The blue bars show the distribution of the L_2-distance for the positive pairs and the orange colored bars for the negative pairs. The brown colored area shows the overlap between the two histograms.

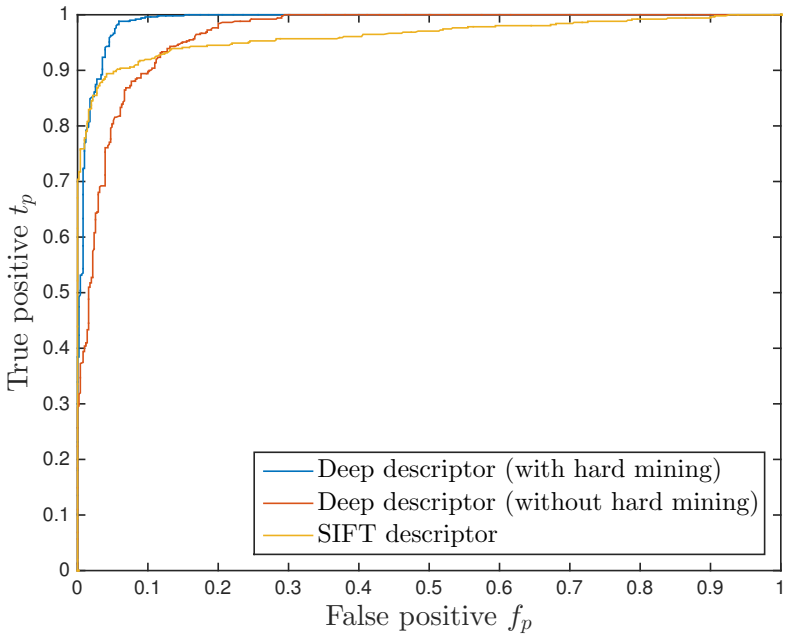


Figure 4.4: ROC-curve of the learned deep descriptor with and without hard mining and SIFT descriptor.

Table 4.3: Validation and test error for classifying network and semantic segmentation (Dilation 10).

Method	Validation error	Test error
Classifying network (without weight decay and weight initialization)	11.17%	11.31%
Classifying network (with weight decay and weight initialization)	10.91%	11.08%
Semantic segmentation (Dilation10)	8.96%	9.73%

4.2 Classifier

The network was trained for 317 epochs (61 815 iterations) and was terminated when the validation error had stabilized. The final validation error obtained was 11.17% and test error of 11.31%. When weight decay and weight initialization was added the performance did improve to a validation error of 10.91% and test error of 11.08%. The semantic segmentation approach, using Dilation10, yields a validation error of 8.96% and test error of 9.73%. The results are summarized in Table 4.3. Example of correctly and incorrectly classified stable and unstable patches are shown in Fig. 4.5.

(a)

(b)

(c)

(d)

Figure 4.5: Example of classified patches. **(a)**: Correctly classified stable patches. **(b)**: Incorrectly classified unstable patches. **(c)**: Incorrectly classified stable patches. **(d)**: Correctly classified unstable patches.

5
Discussion

5.1 Descriptor

The results from the descriptor evaluated on the test set are presented in Table 4.1. From the table one can conclude that the trained descriptor performs better than the SIFT descriptor on urban street scenes when comparing AUROC. Since AUROC is a measurement of the overlap between the positive and negative norm histograms this can be seen in Fig. 4.1, 4.2 and 4.3. The overlap for the trained descriptor using hard mining is small while it is substantially larger when not training with hard mining. From this one can conclude that a hard mining approach implies a significant improvement of the descriptor's performance. The SIFT histogram has two distinct distinguishable peaks but a wide overlap in between which implies that the descriptors have trouble finding a distinct threshold to separate similar patches with non-similar. The result shows the power of a trained descriptor given a dataset containing the transformations to handle. In our case it was rotation, noise and scale, transformations SIFT claims to handle to some extent. By training on the chosen scenarios a trained descriptor can perform better compared to a traditional descriptor. However, notice that SIFT is not specifically compiled for these transformations and not optimized for the performed tests.

From the Cityscapes dataset were 64 002 patches extracted from 21 334 feature points which resulted in a positive set of cardinality $21\,334\binom{3}{2} = 64\,002$ and negative set of cardinality $\sum_{i=1}^{21\,334} 3(64\,002 - 3) = 64\,002(63\,999) = 4\,096\,063\,998$. The huge number of possible pairs makes it unlikely for overfitting and during the training no indications of overfitting were encountered. This was expected since neither Simo-Sierra *et al.* [33] nor Zagoryuko *et al.* [43] encountered any overfitting during their training.

Furthermore, any major conclusions about the descriptor cannot be made since it has not been tested in any applications yet. The motivation behind compiling a trained descriptor came from creating robust maps. A fairer evaluation between the trained and a traditional descriptor would be to test them on, e.g. a SLAM solver. From such comparison a more extent analysis of the trained descriptor can be made.

5.1.1 Creation of Dataset

The created dataset was fairly easy and did not include any hard challenges. From our result one can conclude that the trained descriptor is able to outperform a traditional descriptor despite that the dataset relied on SIFT points and transformations

the SIFT descriptor is invariant against. Furthermore, the dataset was specifically constructed for matched SIFT points. This shows the power of a trained descriptor tweaked for a specific task.

To analyze further the power and capabilities of a trained descriptor, a larger dataset is needed. The compiled dataset contains patches from different seasons but all positive triplets are from the same. Adding additional synthetic transformations, besides the one dealt in this thesis, such as affine transformations is easily done. The dataset should also contain patches from the same feature point but from different point of views and weathers, where the images from the two similar patches were taken from two different places. The required data was, however, not available and creating it from scratch was out the scope for this thesis.

5.1.2 Replacement of L_2 comparison

An L_2-distance between the two descriptors was used as a similarity measurement. Instead of the L_2-distance as similarity measurement it could be replaced by a FCNN, functioning as a decision network. The approach has been used by Zagoruyko [43] and was tried to be replicated, unsuccessfully however. The training was extremely sensitive and converged always to label every pair as similar. The decision network was also trained while the Siamese network was kept unchanged and both including and excluding hard mining. Despite all these approaches and trying to mimic Zagoruyko this was not executed successfully.

5.1.3 Runtimes

The SIFT descriptor is faster than running the trained deep descriptor on either a CPU or a GPU. The convolution layers are more computationally expensive compared to calculating a histogram of gradients at the feature point. The average runtime for the deep descriptor is of the same order of magnitude as the SIFT descriptor when introducing a GPU. This illustrates the essence of executing the computations in the convolutional layers in parallel and carrying out the training on a GPU.

5.2 Classifier

The results from training and evaluation of the feature point classifier are shown in Table 4.3. From the table one can conclude that the semantic baseline method has a slightly better overall performance. This technique managed to classify the feature points as either stable or unstable with a test error of 9.73 %. The first training approach of the classifier yields a test error of 11.31 % and the second where weight decay and weight initialization were introduced the test error was decreased to 11.08%. Further improvements were sought by tweaking different parameters, changing the structure of the FCNN and decreasing the learning rate when training had stagnated. However, they did not increase the network's performance.

While the overall performance was similar for the feature point classifier and the semantic baseline method they differed a lot on feature points placed on large and

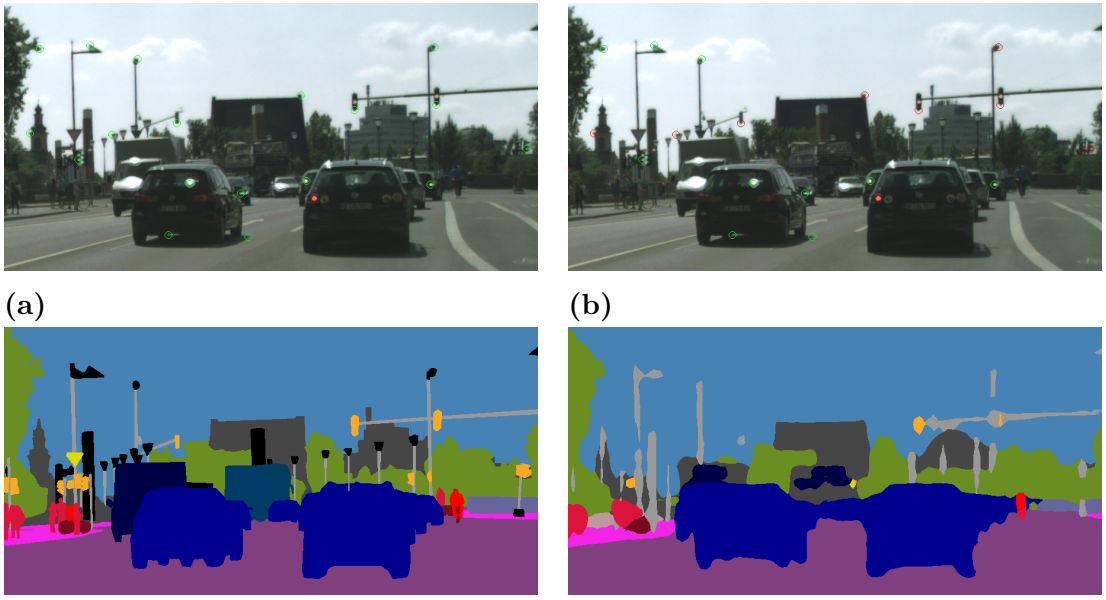


Figure 5.1: Typical situation where the classifier performs better than the semantic baseline method. The green circles correspond to a correct prediction of the feature point and red not. **(a)**: Predictions from the classifier. **(b)**: Predictions from the semantic baseline method. **(c)**: Ground truth annotations. **(d)**: Annotations by Dilation10.

small objects. Fig. 5.1 and 5.2 show the prediction of stable and unstable classes of the classifier and the semantic method as well as the ground truth annotation and the prediction from Dilation10. A green circle around the feature point corresponds to a correct prediction and a red not. Comparing Fig. 5.1a and 5.1b it is concluded that the classifier has no issues with error classifications for small objects such as traffic lights while the semantic method has major problem. Similarly, can the semantic method easily determine larger objects such as buildings, see Fig. 5.2b, while the classifier finds it difficult according to Fig. 5.2a. One reason is the fixed patch size which is suitable for traffic lights from far distance but not from near buildings. One improvement would be to include various scales for each feature point to make the classifier invariant of scale.

The classifier lacks the ability to interact with its neighbors when predicting. A promising development of semantic segmentation is through conditional Markov random fields where adjacent pixels should have similar predictions [22]. This might improve the semantic segmentation on small objects.

5.2.1 Creation of Dataset

The extracted classifications of the patches were decided with the high quality pixel-level annotation of the Cityscapes dataset images. The annotation is made by representing each classified area as a polygon. Since it is likely to detect a feature point between two objects misclassifications did occur. When creating our dataset, one only had to take in consideration the borders between stable and unstable classes.

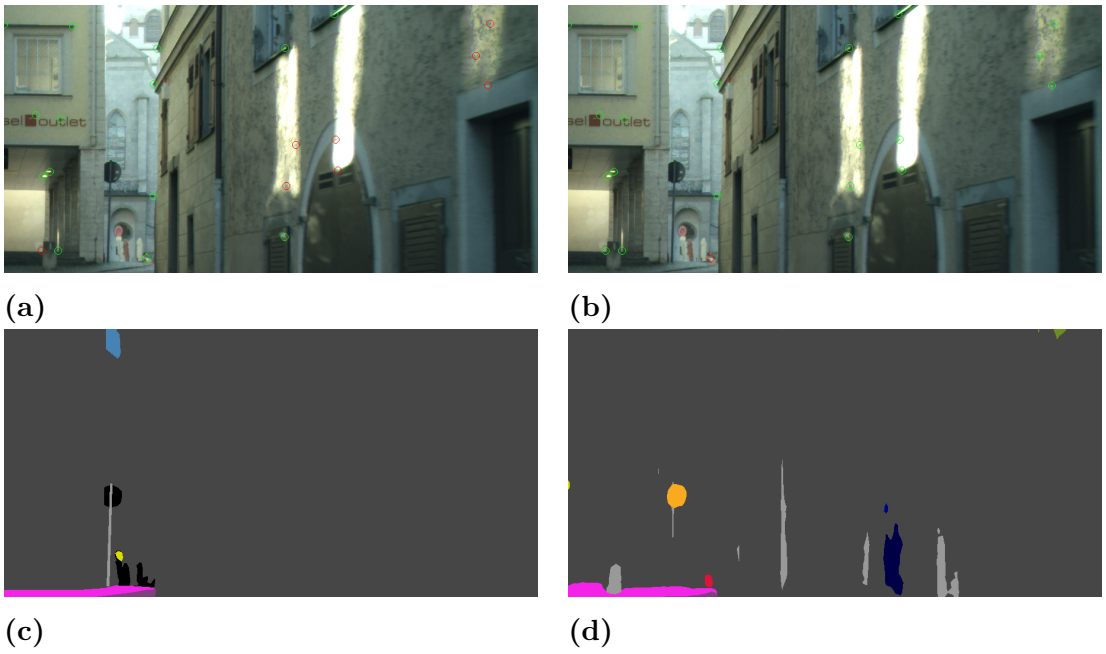


Figure 5.2: Typical situation where the semantic baseline method performs better than the classifier. The green circles correspond to a correct prediction of the feature point and red not. **(a)**: Predictions from the classifier. **(b)**: Predictions from the semantic baseline method. **(c)**: Ground truth annotations. **(d)**: Annotations by Dilation10.

E.g. classifying a feature point as a rider or a cycle, which both are considered to be unstable, does not matter while a rider and a building does. This issue was solved by expanding the annotated area for classes in front of others. E.g. a lamppost with sky around it was extracted since the feature point was always placed on the lamppost and not the sky. The same procedure was applied for pedestrians and vehicles in front of a building. Using these small ad hoc, no mismatches discovered by the human eye were encountered.

When dealing with the approach of classifying feature points as stable or unstable two questions arise. Is the definition of a stable or unstable point representable with annotated images and if so, what defines a stable point? In this thesis a class was defined as stable if it was assumed to be localized every time the scene is observed. An example of such class is a building, while a car or pedestrian clearly is not. However, the separation is not distinct as e.g. commercials on buildings containing humans get classified as unstable, see Fig. 4.5c. Instead of determining the partition of stable and unstable classes in advance the system could learn this. This could be done by driving through the same area multiple times and looking for feature points with similar location and description in all drive-through. Each such localized feature point defines a consistent area suitable for the map. Using this approach, a front and an advertising poster on a building will be treated differently. However, the approach requires the production of such a dataset and methods to match feature points, something that, due to time, was out of the scope for this thesis.

It would be desirable to extend the dataset to be more specific for our task.

Classes as rider and cyclist is not interesting to distinguish from. However, there are classes that should be split into several. The classifier had problems classifying feature points between road surface and vehicles. Furthermore, asphalt is not distinctive and therefore hard to find feature points in. For this reason, road was treated as an unstable class. However, lane marks are preferable features for a map and should be a stable class. The same reasoning applies for the trunk of a tree as a stable class while its leaves and branches are unstable.

5.3 Future Work

The investigation of training a CNN to achieve robust feature descriptors in urban street scenes has only begun. To further continue the investigation a larger dataset is needed, containing not only synthetic transformations but also changes in weather and lighting conditions at the same location. Such a dataset with known transformation between the images does, to my knowledge, not exist. In addition, an interesting area to analyze further is the implication of replacing the L_2-norm with a decision network. Perhaps this approach could separate positive and negative pairs of patches even further. The architecture of the network could also be experimented with. New observations and discoveries occur on a daily basis within the field of CNN, and some might be applicable for the feature descriptor. One approach, proposed by He *et al.* [17], is a deep residual learning framework where the depth of the network increases but not necessarily the complexity. The main reason behind this idea is that as the depth increases the features can be enriched. Another interesting method to try is batch normalization, proposed by Ioffe *et al.* [18]. By normalizing the input to each layer of the network Ioffe *et al.* claim a larger learning rate is possible which implies faster convergence.

An obvious improvement of the trained feature point classifier is to extend the dataset for our purpose. It includes, merging some classes, such as rider and cyclist and splitting some classes, such as road to asphalt and lane marks. To further boost the classifier's performance, training on patches of different scales would be interesting. By letting the classifier train on different scales it might be able to handle close and large objects. The Cityscapes dataset was released February 2016 and new records on the pixel-level semantic labeling task are frequently beaten. The currently best network uses Conditional Random Fields (CRFs) such that adjacent pixels are favorable to be classified as the same [22]. Adding these CRFs might improve classification around edges of objects as well as small objects. It would be interesting to compare such a technique against the feature point classifier.

The descriptor and the classifier has not been evaluated in an application yet. A possible evaluation would be to create four maps, using SLAM, with either a traditional or the trained descriptor and with either the classifier or not. These four maps are evaluated when positioning within the map at a later time with the same setup of descriptor and classifier as when the map was created.

6
Conclusion

In this thesis a feature point descriptor and a classifier, applicable in urban street scenes, using a convolutional neural network (CNN) approach were presented. The problem of representing a feature point in an unambiguous way was tackled through a CNN. The descriptor was trained using a Siamese network with the purpose of differentiating between pairs of similar and non-similar patches, extracted from urban street scenes. A training technique, called hard mining, where only the most difficult pairs affected the network during training was included. By including desirable scenarios, the descriptor ought to handle in the training, it has been shown how the trained descriptor adapts to these. The dataset compiled for this part of the thesis contained scale and rotation transformations and noise adding. The trained feature point descriptor obtained an AUROC score of 0.990 with hard mining and 0.966 without. Both approaches outperformed the traditional SIFT descriptor with an AUROC score of 0.962.

The problem of classifying what kind of object a feature point is placed on was tackled through a CNN. Given a patch around the feature point it was classified as either stable or unstable. A class is considered stable if it is expected to always appear at the same location in a scene, e.g. a building, and unstable otherwise, e.g. a pedestrian. The patch was fed to a pretrained CNN with a deep feature vector as output. This was classified through a fully connected neural network as either stable or unstable. The dataset compiled for this part of the thesis was created by the urban street scenes in Cityscapes dataset and its high quality pixel-level annotations. Given a partition of the stable and unstable classes the best network achieved a test error of 10.91%. This was compared to a model for semantic segmentation, Dilation10, with the set partition of classes. This model achieved a test error of 9.73%. The trained classifier outperformed the semantic approach on small objects but performed worse on large objects.

Bibliography

[1] Brian Ayers and Matthew Boutell. Home interior classification using sift keypoint histograms. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. IEEE, 2007.

[2] HB Barlow, Roo Fitzhugh, and SW Kuffler. Change of organization in the receptive fields of the cat's retina during dark adaptation. *The Journal of physiology*, 137(3):338, 1957.

[3] Ian Bleasdale. Times square 1994, 1994. Licensed under CC BY 2.0. URL: `https://c8.staticflickr.com/3/2837/9408840143_c109caafe9_z.jpg`.

[4] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[5] Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(1):43–57, 2011.

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[7] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Alan J Danker and Azriel Rosenfeld. Blob detection by relaxation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):79–92, 1981.

[11] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[12] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii:

Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012.

[13] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[15] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[16] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL: `http://arxiv.org/abs/1512.03385`.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Ayush Patidar1 Prof Sanjiv Kumar. A review paper on self-driving car's and its applications. 2016.

[22] Guosheng Lin, Chunhua Shen, Ian D. Reid, and Anton van den Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *CoRR*, abs/1504.01013, 2015. URL: `http://arxiv.org/abs/1504.01013`.

[23] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994.

[24] Todd Litman. Autonomous vehicle implementation predictions.

[25] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[26] Krystian Mikolajczyk. *Detection of local features invariant to affine transformations*. PhD thesis, Citeseer, 2011.

[27] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015.

[28] Edwin Olson. Recognizing places using spectrally clustered local matches. *Robotics and Autonomous Systems*, 57(12):1157–1172, 2009.

[29] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[30] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. Institute of Electrical & Electronics Engineers (IEEE), nov 2011. URL: `http://dx.doi.org/10.1109/ICCV.2011.6126544`, `doi:10.1109/iccv.2011.6126544`.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. `doi:10.1007/s11263-015-0816-y`.

[32] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010.

[33] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[35] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, 2015.

[36] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL: `http://arxiv.org/abs/1409.4842`.

[38] Douglas A Thornton, Keith Redmill, and Benjamin Coifman. Automated parking surveys from a lidar equipped vehicle. *Transportation Research Part C: Emerging Technologies*, 39:23–35, 2014.

[39] Sebastian Thrun and John J Leonard. Simultaneous localization and mapping. In *Springer handbook of robotics*, pages 871–889. Springer, 2008.

[40] Jaycil Z Varghese and Randy G Boone. Overview of autonomous vehicle sensors and systems.

[41] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[42] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL: http://arxiv.org/abs/1511.07122.

[43] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[44] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.