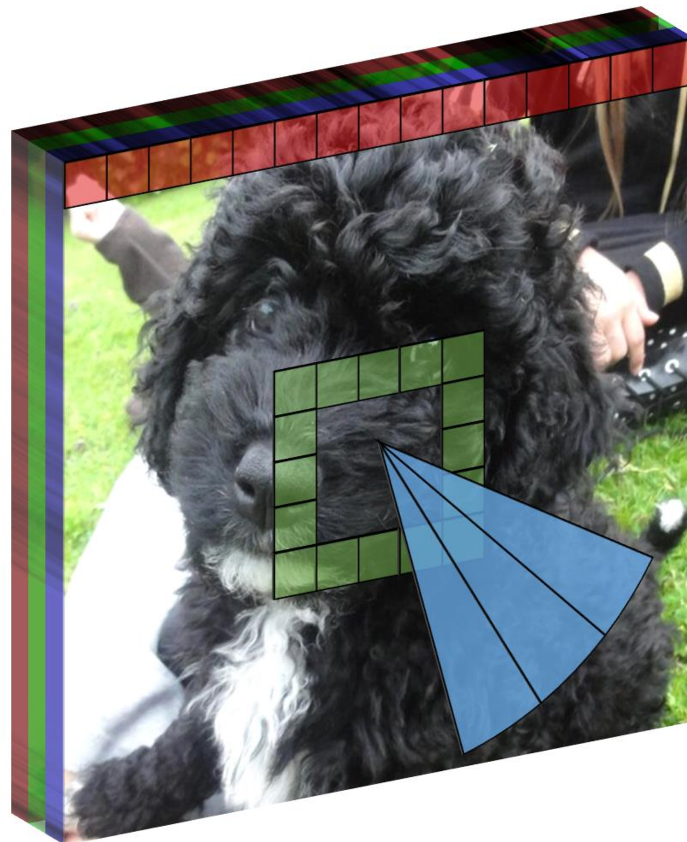

Relative Pose Regression using Non-Square CNN Kernels

Estimation of translation, rotation and scaling between image pairs with custom layers

Master's thesis in System, Control and Mechatronics

JONAS KARSTRÖM
ÖRJAN LANDGREN

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:23

Relative Pose Regression using Non-Square CNN Kernels

Estimation of translation, rotation and scaling between image pairs with custom layers

JONAS KARSTRÖM
ÖRJAN LANDGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2020

Relative Pose Regression using Non-Square CNN Kernels
Estimation of translation, rotation and scaling between image pairs with custom layers
JONAS KARSTRÖM
ÖRJAN LANDGREN

© JONAS KARSTRÖM, ÖRJAN LANDGREN, 2020.

Supervisor: Robert Brenick, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:23
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The RGB image on the cover depicts the cute dog Kira and three specially designed CNN kernels, see section 3.3 for more information.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Relative Pose Regression using non-square CNN kernels
Estimation of translation, rotation and scaling between image pairs with custom layers
JONAS KARSTRÖM
ÖRJAN LANDGREN
Department Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Localisation is a research field where handcrafted and complex engineering methods have so far given the best results. However, a general trend in computer science is that data-driven approaches often outperform classical methods. These data-driven approaches have been enabled by better high-resolution range sensors and vast amounts of data. End-to-end deep neural network approaches have much better scalability, but the state of the art localisation networks still use designs originally developed for image classification. These are necessarily not the best-suited design, as they aim to achieve a different task than relative pose regression.

We show that custom input layers designed to predict translation, rotation or scaling between images are a possible solution. The networks tested in this thesis give good results, but do not outperform a baseline neural network with the same design as a neural network initially made for image classification.

Keywords: Convolutional Neural Network, Kernel, Layer, Localisation, Machine Learning, Relative Pose Regression, Visual Odometry.

Acknowledgements

We would like to take the opportunity to thank our examiner Peter Forsberg and our supervisor Robert Brenick, for their help and continuous support during our master thesis. We would also like to thank CPAC Systems AB that has taken good care of us during the master thesis. We also want to thank Elin Ekdahl, Hulda Landgren, Oscar Lundström and Amanda Fransson for feedback on our report. Finally, we would like to thank Kira the Perro de agua español for posing in many images.

Jonas Karström and Örjan Landgren, Gothenburg, June 2020

Thesis supervisor: Robert Brenick
Thesis examiner: Peter Forsberg

Abbreviations

ACV	-	Autonomous Commercial Vehicles
APR	-	Absolute Pose Regression
BA	-	Bundle Adjustment
CNN	-	Convolutional Neural Network
DCNN	-	Deep Convolutional Neural Network
DNN	-	Deep Neural Network
DoF	-	Degrees of Freedom
DR	-	Dead Reckoning
FCL	-	Fully Connected Layer
GNSS	-	Global Navigation Satellite System
ICP	-	Iterative Closest Point
IMU	-	Inertial Measurement Unit
IR	-	Image Retrieval
LiDAR	-	Light Detection And Ranging
ML	-	Machine Learning
MSE	-	Mean Squared Error
NN	-	Neural Network
PTAM	-	Parallel Tracking And Mapping
R-CNN	-	Rotating-Convolutional Neural Network
RGB	-	Red, Green, Blue
RGB-D	-	Red, Green, Blue, Depth
ReLU	-	Rectified Linear Unit
RNN	-	Recurrent Neural Networks
RPR	-	Relative Pose Regression
S-CNN	-	Scaling-Convolutional Neural Network
SLAM	-	Simultaneous Localisation and Mapping
T-CNN	-	Translating-Convolutional Neural Network
VO	-	Visual Odometry
V-SLAM	-	Visual Simultaneous Localisation and Mapping

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Related work 2
1.1.1 Visual odometry 2
1.1.2 Visual simultaneous localisation and mapping 2
1.1.3 Image retrieval 3
1.1.4 Handcrafted or data-driven approach 3
1.1.5 Pose regression 3
1.2 Problem formulation 4
1.2.1 Purpose 5
1.2.2 Scope 5
1.3 Report outline 5

2 Theory **7**
2.1 Digital images 7
2.2 Neural networks 7
2.2.1 Neurons 8
2.2.2 Activation functions 8
2.2.3 Layers 9
2.2.4 Kernels 11
2.3 Training a neural network 12
2.3.1 Dataset 13
2.3.2 Loss function 13
2.3.3 Backpropagation 13
2.3.4 Optimiser 16
2.3.5 Overfitting 16

3 Method **19**
3.1 Development environment 19
3.2 Dataset 19
3.3 Kernel shapes 22
3.3.1 Translation 22
3.3.2 Rotation 23
3.3.3 Scaling 25

3.4 Networks . . . 25
3.4.1 Baseline CNN . . . 25
3.4.2 Translation network (T-CNN) . . . 26
3.4.3 Rotation network (R-CNN) . . . 26
3.4.4 Scaling network (S-CNN) . . . 27
3.4.5 Training . . . 27
3.4.6 General design . . . 28

4 Results **29**
4.1 Translation . . . 29
4.2 Rotation . . . 31
4.3 Scaling . . . 32

5 Discussion **35**
5.1 All networks converge . . . 35
5.2 Comments on results . . . 35
5.2.1 Translation . . . 35
5.2.2 Rotation . . . 36
5.2.3 Scaling . . . 36
5.3 Image resolution . . . 37
5.4 Implementation . . . 37
5.4.1 Rotation layer . . . 37
5.4.2 Network layout . . . 39
5.4.3 Network comparison . . . 39

6 Conclusion **41**
6.1 Future work . . . 41
6.1.1 Relative pose regression . . . 42
6.1.2 Absolute pose regression . . . 42

A Appendix **I**
A.1 Examples from the first datasets . . . I
A.2 First testing of R-CNN and baseline CNN on 28x28 images . . . II
A.3 Translation along only Y-dimension . . . IV

List of Figures

2.1 RGB image . . . 7
2.2 R-channel . . . 7
2.3 G-channel . . . 7
2.4 B-channel . . . 7
2.5 Grayscale image . . . 7
2.6 ReLU. . . . 9
2.7 Sigmoid. . . . 9
2.8 Log Sigmoid. . . . 9
2.9 Example of a fully connected network. This fully connected network has two inputs, one output and three hidden layers in between. . . . 9
2.10 Example of a 2D Convolution operation with a stride of 1 with no bias 10
2.11 Example of a CNN. The input to this network is an RGB image with 100x100 pixels. The network contains convolutional layers, pooling layers as well as FCL. . . . 10
2.12 Example of a Max-Pool operation with a size (1,2,2) and stride of 2. . 11
2.13 Figure 2.1 with padding size one. . . . 11
2.14 Input image from Figure 2.13 through a square kernel CNN. . . . 11
2.15 An example of how an unusual kernel shape can look. This figure depicts a dialated kernel that strides over an image. . . . 12
2.16 Explanation of manual tuning of a NN by xkcd.com. . . . 12
2.17 Example of a NN, one can see how the first layer A affects the last layer D. . . . 15
2.18 Example of early stopping. . . . 17

3.1 Example images from the MNIST dataset. Consisting of images with resolution 28x28, depicting handwritten digits in grayscale. . . . 20
3.2 Examples of images from the STL-10 dataset. Consisting of colour images with resolution 96x96. . . . 20
3.3 10 example image pairs from the constructed translation dataset. . . 21
3.4 10 example image pairs from the constructed rotation dataset (before being cropped to 48x48). . . . 21
3.5 10 example image pairs from the constructed scaling dataset. . . . 22
3.6 Example of how a 3x3 square kernel stride over an input image. Note that the actual pixel size of the image is much smaller than the white example pixels. . . . 22

3.7 Example of how the first layer of the T-CNN processes the input image. Note that the actual pixel size of the image is much smaller than the white example pixels. . . . 23
3.8 Input for regression along the vertical axis. . . . 23
3.9 Input for regression along the horizontal axis. . . . 23
3.10 Example of how the first layer of the R-CNN processes the input image. Note that the actual width of the kernel is smaller than in the example. . . . 24
3.11 Example of how a kernel strides over an image. Image size (28x28), with a kernel of 48 pixels, rotated around the image in 16 steps. . . . 24
3.12 Example of how the S-CNNs first layer process the input image. Note that the actual pixel size of the image is much smaller than the white example pixels (as for all following examples). . . . 25
3.13 Visualisation of the baseline CNN network. . . . 26
3.14 Visualisation of the T-CNN network. . . . 26
3.15 Visualisation of the R-CNN network. . . . 27
3.16 Visualisation of the S-CNN network. . . . 27

4.1 Translation Baseline CNN histograms. One histogram for each translation direction. . . . 30
4.2 Translation T-CNN histograms. One histogram for each translation direction. . . . 30
4.3 Rotation CNN histogram. . . . 31
4.4 Rotation R-CNN histogram. . . . 32
4.5 Scaling CNN histogram. . . . 33
4.6 Scaling S-CNN histogram. . . . 33

5.1 Example of an alternative, siamese T-CNN network. . . . 36
5.2 Images showing that the interpretation of the inputs to a circle sector filter at two different angles is not trivial. . . . 38

6.1 Example of a car with an upward and a forward facing camera. . . . 42

A.1 10 example image pairs from the constructed rotation dataset. . . . I
A.2 10 example image pairs from the constructed translation dataset. . . . I
A.3 10 example image pairs from the constructed scaling dataset. . . . I
A.4 Results on the rotation test set with one of the first models of the R-CNN using a preprocessing implementation for the specially designed rotational layer. . . . II
A.5 A first baseline CNN performance on the rotational test set (constructed from the MNIST dataset). . . . III
A.6 Resulting histogram for the baseline CNN, translation along only Y-dim. . . . IV
A.7 Resulting histogram for the T-CNN, translation along only Y-dim. . . . V

List of Tables

2.1 A simplified step by step description of how to train a NN. 13

4.1 Results from the translation tests. The values in the table is for translations along both directions. 29
4.2 Results from the rotation tests 31
4.3 Results from the scaling tests 32

A.1 Results from the translation test (only along Y-dim). IV

1
Introduction

Since the turn of the century, the market for automated commercial vehicles (ACVs) has grown at an increasing rate. There is still a large potential for expansion in the ACV market. New and smarter ACVs are on the horizon, from small automated cars to automated hauling trucks for mining operations.

To enable this kind of expansion, robust, reliable, and accurate navigation is necessary for safe deployment in complex environments. Navigation consists of many sub-problems, one of them is localisation, which will be the focus of this thesis.

Examples of ways to measure movement, that can be used to do localisation are:

- *Wheel-odometry* uses motion sensors that measure the rotation of the vehicle's wheels. From these sensor readings, the vehicle's movement over time is estimated.
- *Inertial measurement units* (IMUs) consists of accelerometers, gyroscopes and sometimes magnetometers. IMUs are commonly used to estimate acceleration, angular velocity and angle.
- *Light detection and ranging* (LiDAR) uses lasers to measure the distance to objects in the environment, this can be used to estimate the change in position.
- *Visual odometry* (VO) is a method to estimate the ego-motion of a camera, which is the change in position and rotation of where two consecutive photos were taken.

Dead reckoning (DR) is the process of calculating the current position given knowledge of the movement from an initial position. DR is a relative localisation method and it suffers from accumulative error. Wheel odometry, IMU and VO are used to estimate a relative change in pose and can be used for DR.

To be able to synchronise the movement methods such as *simultaneous localisation and mapping* (SLAM) can be used. SLAM both localise and simultaneously map a previously unknown environment. The map is continuously updated to make the localisation globally consistent.

To update the map the SLAM algorithms often perform loop closing and *bundle adjustment* (BA) which, if successful, gives a globally consistent map. This enables global localisation with sensors that are made for relative localisation.

Another strategy for doing localisation is the *global navigation satellite systems* (GNSS), which uses satellites to triangulate the position of a GNSS receiver. GNSS is an absolute localisation method with a geocentric frame of reference and does not suffer from accumulative errors.

1.1 Related work

The area of mobile robotics has interested and engaged researchers all over the world for decades [1]. To grasp a rapidly changing research field can be a difficult task. Reading an overview such as [1] or [2] is recommended for the uninitiated.

1.1.1 Visual odometry

Visual odometry, or VO, as previously stated, is the task of estimating the relative pose of two cameras. VO is a part of the bigger field called “Structure from motion”, which purpose is to estimate a relative pose and the surrounding environment. The most common approach to do VO includes the following steps [3]:

1. From an image sequence extract features
2. Match and track the features across cameras and between consecutive frames
3. Do motion estimation

There are many ways to extract features, using detectors to find them and descriptors to describe them. Some well-known combined feature detectors and descriptors are SIFT [4], SURF [5] and ORB [6]. ORB combines the BRIEF [7] descriptor and FAST [8] feature detector. The matched features are triangulated to estimate the ego-motion and are optimised using a robust fitting algorithm such as RANSAC [9]. BA [10] is often used to further optimise the camera trajectory and the map representation if a SLAM system is used.

1.1.2 Visual simultaneous localisation and mapping

Visual simultaneous localisation and mapping (V-SLAM) is SLAM using images to do VO and reconstruct the map. An example of a V-SLAM system is *parallel tracking and mapping* (PTAM) by Klein and Murray [11]. This work was groundbreaking, but still had limitations, one of them was that it only worked on a small scale. A significant improvement to PTAM was made in the feature-based SLAM system ORB-SLAM [12]. As the name suggests, it is based on the ORB features. An updated version ORB-SLAM2 [13] was presented 2017 and at the time of release, it was the best performing open-source V-SLAM system. ORB-SLAM2 can work with either monocular, stereo or RGB-D input. Currently, the top tier V-SLAM methods are feature-based, although there are some exceptions such as BAD-SLAM, bundle adjusted direct RGB-D SLAM [14]. In the BAD-SLAM article, they state that direct-SLAM, as a contrary to feature-based SLAM, is sensitive to rolling shutter effect and synchronisation error of the RGB and depth input. To be able to fairly compare different SLAM methods, a new benchmark was created where these problems were solved in the hardware. A global shutter was used and the RGB and depth input synchronised. When BAD-SLAM was tested on this new benchmark and compared with the previous state of the art, such as ORB-SLAM2, BAD-SLAM outperformed its feature-based counterparts.

1.1.3 Image retrieval

Image retrieval (IR) is, in a very broad sense, a system for searching, browsing and retrieving images from a large image database. In the context of *neural networks* (NN), explained in section 2.2, IR is primarily used to do place recognition [15, 16, 17, 18]. IR can also be used in the localisation task. IR is used to roughly estimate the pose by using the pose of the most similar image or by weighing together a number of the best matches.

1.1.4 Handcrafted or data-driven approach

Localisation is, as previously mentioned, a research field that has been very active for quite some time. Handcrafted, complex engineering methods are what still perform best by far. However, it is impossible to know if it always will be like this. A general trend in computer science is that data-driven approaches are growing and in multiple fields outperforms classical methods. With new and better high-resolution range sensors, vast amounts of data are made available. This could, of course, change the field since classical approaches such as *iterative closest point* (ICP) does not scale well with increased frame rates and data flows. ICP also has a risk of divergence if the initially estimated pose is far from the true pose. Therefore, new ways of doing localisation and mapping are explored.

1.1.5 Pose regression

The first end-to-end *deep neural network* (DNN) that did *absolute pose regression* (APR) is called PoseNet [19]. PoseNet is based upon GoogLeNet [20], a deep convolutional neural network (DCNN) that was designed for image classification. Transfer learning (on image classification) is used, to drastically decrease the amount of training data needed, the last few layers are exchanged to let the net train on the 6 *degrees of freedom* (DoF) pose for images taken in central Cambridge. An online demonstration is presented at their project webpage [21]. PoseNet takes as input an image and gives as output a 6 DoF pose. Given this approach, the network naturally needs to be trained on image input that is from the same area as the test input.

Since PoseNet, many new neural net approaches have been published which improve upon PoseNet. VLocNet++ [22] has a similar approach but is a bit more advanced. Instead of one single network, it combines four networks. The four networks share weights and information. Two networks are trained to give a relative pose estimation between the current and the previous image. One network is trained for the global pose and one network is trained to do image segmentation. The idea behind this multitask learning for: (1) odometry estimation, (2) 6 DoF visual localisation and (3) semantic segmentation, is to exploit interdependence's in these tasks for mutual benefit.

One more NN approach that should be presented due to its interesting design is attention-based global pose estimation network [23]. Attention was first a method to deal with some of the shortcomings of *recurrent neural networks* (RNN) [24] but

was then shown to work on its own in the paper "Attention is all you need" [25]. The method presented in this article [23] used a straight forward DCNN to do *relative pose regression* (RPR) between two consecutive images and then used an attention network to estimate the global pose from the relative poses. The pose was however only estimated in 3 DoF, two for translation and one for rotation.

Two master theses were presented at Chalmers 2019. One by Brenick and Bastås [26], which built upon PoseNet, with added depth information and one by Linderoth and Lundqvist [27] which aimed at doing relative pose regression between LiDAR scans and a known map. A big part of their work consisted of letting CNNs train on 2D LiDAR intensity images, one from the LiDAR sweep and one cut-out from the prebuilt map.

Current limitations

Convolutional neural networks (CNNs) have proven successful in many fields. The usage of CNNs will probably increase in the future, also in the localisation field. However, in the article "Understanding the Limitations of CNN-based Absolute Camera Pose Regression" [28] by Sattler *et al.*, it is shown both through theoretical explanation and experimental tests that CNN-based approaches do not generalise beyond the training data. The authors claim that:

"A key result is that current approaches do not consistently outperform a handcrafted image retrieval baseline. This clearly shows that additional research is needed before pose regression algorithms are ready to compete with structure-based methods."

This article and another one, by the same authors on the same topic; "To Learn or not to learn: Visual Localisation from Essential Matrices" [29] pinpointed the need for more basic research for learning-based methods to be able to compete with structure-based methods in the future.

1.2 Problem formulation

As stated above, using various types of CNNs to solve the localisation task has thus far proven unsuccessful compared to structure-based methods. The performance is closely related to IR. There is a wide variety in how the networks are put together, but all CNN methods we have come across heavily depend on networks designed for image classification. These methods have the same key building block: the convolutional layers, that interpret the input images. It is also common to use transfer learning from image classification data. With this in mind, it is not at all surprising that the methods perform similar to IR. We might then ask ourselves if designs developed for image classification is the best approach for pose regression? Is it possible that localisation differs too much from classification and that there could be other, still undiscovered, ways to use CNNs when it comes to image localisation? Could we design a new kind of dedicated layer for CNNs that will enhance the performance?

In most CNN networks [30, 31, 32, 33], the kernel has the shape of a square, with common sizes of 1x1 3x3, 5x5, see the explanation of CNNs and kernels in the following theory chapter in Figure 2.2.3. This type of layer has proven to give good performance in image classification tasks [31, 32, 33], but would it not be more efficient to have a convolution layer that has the shape and stride that fit the localisation problem better? Therefore we will make specific layers that are designed to analyse the image along the axis of change, to efficiently find rotation, translation, and scaling of objects in images.

1.2.1 Purpose

The purpose of this thesis is to find out if custom shapes for kernels, and stride along the axis of interest, in CNNs can give better result than “classical” CNN in the RPR task. More specifically, the task is to develop three different types of layers to be used for scaling, translation, and rotation regression respectively, and compare these to a CNN with regular square kernels, as used in a normal classification network [30]. The work is in some ways similar to that of Linderoth and Lundqvist [27], in the way that both aim at doing NN regression to estimate the rigid transformation, translation, rotation, and (in our case) also scaling. The rigid transformation is to be estimated from two images. In the RPR case, these two images are the current and the previous image, in the APR case, it is the current image and a map cut-out. Our work is different since instead of using various kinds of “standard” CNN designs, we build new network layers that, as far as we know, have never been tried before.

1.2.2 Scope

The scope of the thesis is to design new layers for use in CNNs and to validate the concept. This will hopefully inspire further research in the same direction, to make pose regression that is faster, computationally lighter and at the same time more accurate.

Due to the limited time frame, the following limitations have been made:

- Only RGB images will be used, and the resolution before pre-processing will be limited to 96x96 pixels
- Localisation will be relative between two RGB images of the same size, not necessarily gathered from a localisation situation
- The relative position of the two images will be calculated by addressing scaling, translation and rotation of the images separately

1.3 Report outline

This thesis consists of six chapters, the first one being this introduction. In the next chapter, the theory is presented. The third chapter describes the methods we have used and developed, and thereafter the results are presented in the fourth chapter. The results are discussed in the fifth chapter before the final chapter where the conclusions are drawn.

2

Theory

In the following section, a collection of theory, necessary to understand the work of this thesis is explained.

2.1 Digital images

The two primary methods to describe images in a digital format as vector images and as bitmap images. A vector image is an image that consists of lines and curves that form different geometric shapes. A bitmap image consists of one or multiple finite-sized matrices where the matrices together represent an image. Two very common methods to encode images in the bitmap format is as an RGB image in Figure 2.1, with the red channel in Figure 2.2, green channel in Figure 2.3 and the blue channel in Figure 2.4 colour channel or as a grayscale image in Figure 2.5.

Figure 2.1: RGB image

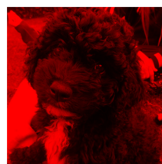

Figure 2.2: R-channel

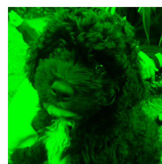

Figure 2.3: G-channel

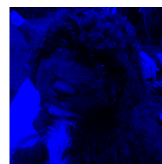

Figure 2.4: B-channel

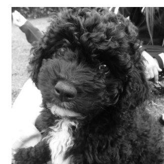

Figure 2.5: Grayscale image

2.2 Neural networks

A *neural network* (NN), in the context of machine learning (ML), can be described as a function approximator, for a given input gives a specific output, $z_j^i = f_j^i(a)$. The name neural network comes from neurobiology. Inspired by, the understanding of, how neuron are connected in the human nervous system a whole area of research has erupted. A NN is made by stacking neurons together in layers, and stacking multiple layers after each other builds a NN, see Figure 2.9. NNs with many layers after each other are referred to as *deep neural networks* (DNN) which have demonstrated to be very powerful in several fields. Examples of areas were DNN has proven to perform well are image recognition, scene understanding and language processing.

2.2.1 Neurons

The most crucial building block of a NN is the neuron. A neuron consists of weights, a bias and a summation. All inputs to the neuron are changed in correspondence with the weights. The weighted inputs are summed together with the bias to form the output of the neuron. Neurons are often stacked together to form layers as described in subsection 2.2.3, neurons can also perform non-linear approximations if paired together with activation functions which are explained in subsection 2.2.2. Mathematically a neuron can be described as a weighted sum, see Equation 2.1, where $\mathbf{a}$ is the input vector which contains all neurons or inputs in its receptive field, b is a scalar bias and z is the scalar output of the neuron. Different outputs can be given by changing the input, bias or weights.

$$z = \sum_{k=1}^{N} w_k a_k + b \tag{2.1}$$

2.2.2 Activation functions

An *activation function* is a nonlinear function that is used between layers in a NN. Activation functions enable a NN to give non-linear outputs a from linear inputs z. The usage of activation functions in NNs enable approximations of non-linear systems. Commonly used activation functions are the rectified linear unit (ReLU), sigmoid and log sigmoid [34]. The equations for these are Equation 2.2, Equation 2.3, and Equation 2.4 where z is the input to the activation function and a is the output. Descriptive plots can be found in Figure 2.6, Figure 2.7, and Figure 2.8. The activation function that is used in the networks, in this thesis, is the log sigmoid function.

$$a = max(0, z) = \frac{|z| + z}{2} \tag{2.2}$$

$$a = \frac{1}{1 + e^{-z}} \tag{2.3}$$

$$a = \ln\left(\frac{1}{1 + e^{-z}}\right) \tag{2.4}$$

The log sigmoid function takes an input and maps it to values between $-\infty$ and 0. The mathematical expression for the log sigmoid function is presented in Equation 2.4. A figure of the output can be seen Figure 2.8. One strength with the log sigmoid is that the derivative is continuous.

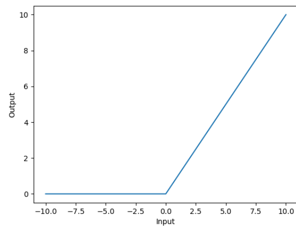

Figure 2.6: ReLU.

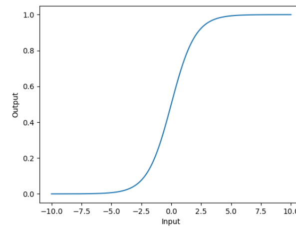

Figure 2.7: Sigmoid.

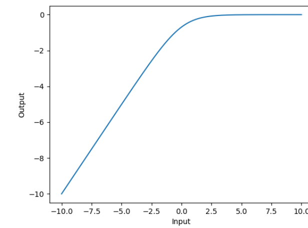

Figure 2.8: Log Sigmoid.

2.2.3 Layers

In the field of supervised learning, a layer maps an input vector to an output vector. A layer that is neither the input nor the output of the system is often referred to as a hidden layer. This mapping can include using a function that has pre-determined output or use trainable weights. Two common layers that use trainable weights are the fully connected layer and the convolution layer, while the pooling layers and activation layers have generally no trainable weights.

Fully Connected Layer

A *fully connected layer* (FCL) is a layer that maps each input neuron with a trainable weight for each output neuron. In Equation 2.5 described how the output is composed of the sum of the weighted inputs and biases in the i:th FCL. An FCL takes a 1D vector of size N and gives an output vector of M with $N \cdot M$ number of weights and M biases.

$$z_j^i = \sum_{k=1}^{N} w_{j,k}^i a_{j,k}^{i-1} + b_j \tag{2.5}$$

$$j \in [1, M] \tag{2.6}$$

A network with only fully connected layers is called a fully connected network or sometimes, a multilayer perceptron, a trivial example of this is presented in Figure 2.9.

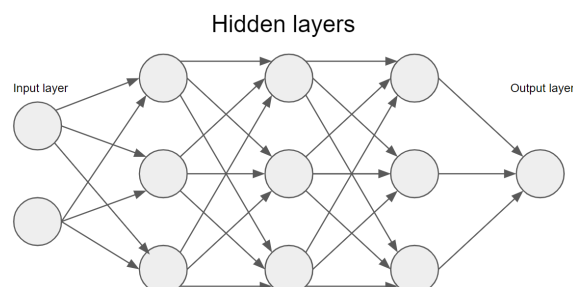

Figure 2.9: Example of a fully connected network. This fully connected network has two inputs, one output and three hidden layers in between.

2D Convolution Layer

The 2D convolution layer is the most central part of any CNN. A 2D convolution layer takes a 3D input of shape (C,X,Y) and produces an output of size ($\tilde{C}$,$\tilde{X}$,$\tilde{Y}$). A kernel is a 3D array of weights with the shape of (C, $\bar{X}$, $\bar{Y}$). The kernel convolves over the second and third dimension of the input array which produces the given output size. Figure 2.10 illustrates how an input of shape (1,5,5) can be processed by a convolution layer with a kernel of shape (1,3,3). This gives an output of shape (1,3,3), the operation between the kernel and the area of convolution is a dot product. By adding more kernels, one can increase the number of output channels $\tilde{C}$.

$$
\begin{bmatrix} 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & -7 & 2 \\ 6 & -8 & 2 \\ 6 & -9 & 3 \end{bmatrix}
$$

Figure 2.10: Example of a 2D Convolution operation with a stride of 1 with no bias

A network which contains at least one convolutional layer is called a *convolutional neural network* (CNN), an example of a CNN is presented in Figure 2.9.

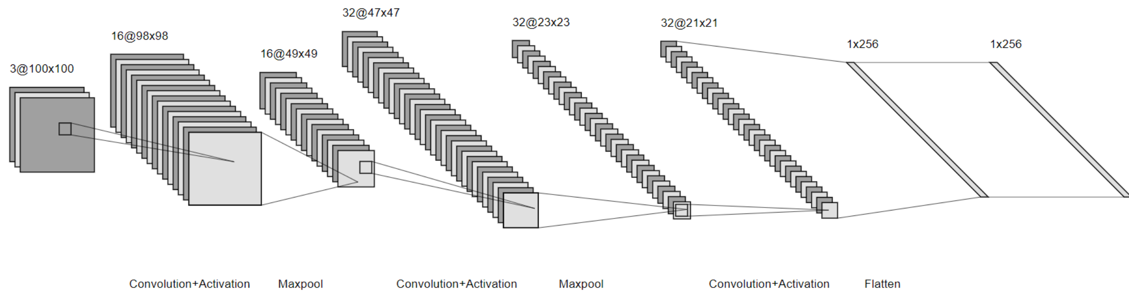

Figure 2.11: Example of a CNN. The input to this network is an RGB image with 100x100 pixels. The network contains convolutional layers, pooling layers as well as FCL.

Pool Layer

A *pooling layer* has a lot in common with the convolution layer. It operates on 3-dimensional matrices and has a set size of (C,$\bar{X}$,$\bar{Y}$). Generally, pooling layers have no trainable weights. It takes the values of the current receptive field, performs an operation, and gives an output of dimension (C,1,1) for every step. Pooling layers are often used to reduce dimensionality in-between convolution layers. The operation is often performed separately for each channel. A commonly used pooling technique is to output the maximum values from each input channel, often referred to as max-pooling. An example of max-pooling is in Figure 2.12.

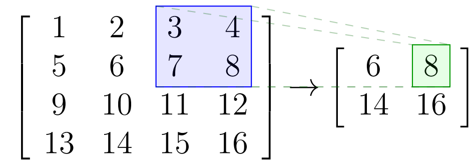

Figure 2.12: Example of a Max-Pool operation with a size (1,2,2) and stride of 2.

2.2.4 Kernels

A kernel is a collection of weights that are cross-correlated with parts of the input to a NN layer, to extract relevant information and forward it to the next layer.

There are different ways to design and use kernels. In the 1D convolution case, the kernel is a vector, this can be used for example in networks that analyse time-series. 1D convolutions can also be used to reduce the number of parameters in the 2D convolution case, using something called separable convolution. In spatial separable convolution 1D kernels are used first along one dimension and then along the other dimension over the output from the first 1D convolution. An example of convolution with a 2D kernel was shown in Figure 2.2.3. There are also 3D convolutions where the input is 4D and the kernel strides along 3 dimensions, but it is not as common. The most common kernel is the 2D square sized kernel, however custom kernels can be designed into arbitrary shape and size.

A visual explanation of an image with padding of size one can be found in Figure 2.13. Here the grey boxes around the image represent a pixel in each direction. This gives the image a resolution of (x+2, y+2), and also increases the output size of the CNN in Figure 2.14 by two in each direction, compared with if the same operation would have been performed without padding.

Figure 2.13: Figure 2.1 with padding size one.

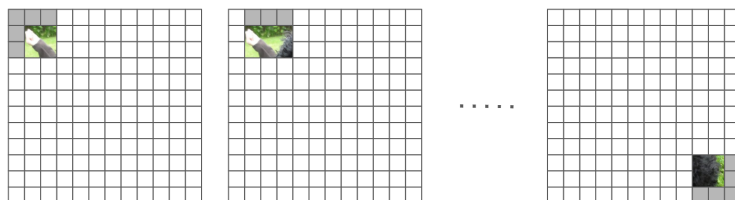

Figure 2.14: Input image from Figure 2.13 through a square kernel CNN.

An example of a kernel with a special shape is the dilated kernel shape which takes the form of a square but with empty spaces in-between itself, this is visualised in Figure 2.15.

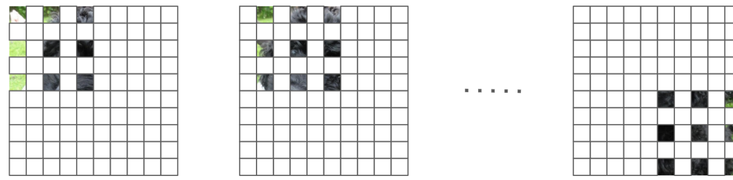

Figure 2.15: An example of how an unusual kernel shape can look. This figure depicts a dialated kernel that strides over an image.

In both Figure 2.14 and Figure 2.15 the stride is one. The term stride, in the ML context, comes from data structures where the expression “to stride” along an array is used. In the same way, here the kernel stride along the horizontal and vertical axis of the input space. In the examples mentioned the output shape would be cut in half in both directions if the stride would change to two, by a third if the stride would be three and so forth.

2.3 Training a neural network

The meaning of “training” a NN is to improve the NNs ability to make better predictions given a set of data. A network is trained by tuning the weights and bias. In theory, a NN could be tuned manually, see Figure 2.16. In practice, this task can escalate quickly, since a “small” NN often have a couple of thousand to millions of weights and biases. To tune a NN, better methods are needed. This section will explain some methods that are used to make tuning more efficient. In Table 2.1 a summary of how to train a neural network is displayed.

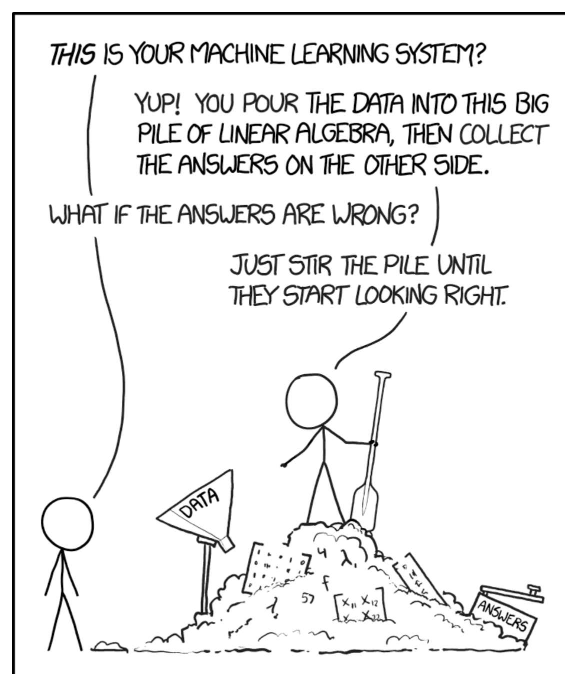

Figure 2.16: Explanation of manual tuning of a NN by xkcd.com.

Step	Action	Section
1	Choose or make a dataset	2.1, 3.2
2	Choose or design a NN	2.2
3	Restrucure the dataset in smaller batches	2.3.5
4	Calculate loss from NN and training set	2.3.2, 2.3.4
5	Do backpropagation to change weights	2.3.3
6	Calculate loss from NN and validation set without dropout	2.3.5
7	If validation loss has decreased, return to 3	
	If validation loss has increased, stop training	2.3.5
8	Calculate loss on test set to assess performance	

Table 2.1: A simplified step by step description of how to train a NN.

2.3.1 Dataset

Problem-solving using NNs is a data-driven approach. In supervised learning, the dataset contains a source (X) and target (y) which come in pairs. In an image classification dataset, the source could contain an RGB image of a dog and the corresponding target would be the string "dog". A dataset is usually divided into a training, a validation and, a test set. As described in Table 2.1, both the training set and validation set are used during training. The training set is used to update the weights and biases, the validation set is used to make sure that the network makes better predictions over the epochs and does not overfit. When a NN has finished the training, the performance is tested on the previously unseen test set.

2.3.2 Loss function

A loss function or cost function is used to measure how successful or unsuccessful an estimation is concerning the desired outcome. In the context of a NN, the loss function is used to compare the desired output y_i to the NNs predicted output $\hat{y}_i$. In regression networks, the *mean squared error* (MSE) loss is often used. As described in Equation 2.8 and as the name suggests, the MSE takes the mean of the squared error of the true value y and the predicted value $\hat{y}$. A strength with MSE loss is that large errors give a much larger loss compared to small errors.

$$error_i = (y_i - \hat{y}_i) \tag{2.7}$$

$$Loss = \frac{1}{n}\sum_{i=1}^{n} error_i^2 \tag{2.8}$$

2.3.3 Backpropagation

Backward propagation of errors or *backpropagation* is an algorithm that calculates how the weights and biases respectively affect the loss of a specific input-output pair. To exemplify this, in a NN with L number of layers, a specific layer is denoted as the i:th layer. In each layer, the respective outputs are associated with weights and bias. In the i:th layer, the j:th output is denoted as z_j^i. The associated weights

are w^i_j and the bias is b^i_j. A specific weight, for example the k:th weight is denoted as $w^i_{j,k}$. Between the layers there is always an activation function which is specified as σ. The activation function maps the outputs from the $(i-1)$:th layer to the i:th layer as in Equation 2.11.

$$a^i_j = \sigma\left(z^i_j\right) \tag{2.9}$$
$$z^i_j = w^i_j a^{i-1}_j + b^i_j \tag{2.10}$$
$$a^i_j = \sigma\left(w^i_j a^{i-1}_j + b^i_j\right) \tag{2.11}$$

To calculate how the k:th weight affect the j:th output in the i:th layer, the chain rule can be applied as in Equation 2.12.

$$\frac{\partial a^i_j}{\partial w^i_{j,k}} = \frac{\partial a^i_j}{\partial z^i_j}\frac{\partial z^i_j}{\partial w^i_{j,k}} \tag{2.12}$$

It is trivial to continue to expand the way of writing equations as in Equation 2.11, where the weights in the $i-1$:th layer affect the output in the i:th layer as described in Equation 2.13-2.18

$$a^i_j = \sigma\left(z^i_j\right) \tag{2.13}$$
$$a^{i-1}_j = \sigma\left(z^{i-1}_j\right) \tag{2.14}$$
$$z^i_j = w^i_j a^{i-1}_j + b^i_j \tag{2.15}$$
$$z^{i-1}_j = w^{i-1}_j a^{i-2}_j + b^{i-1}_j \tag{2.16}$$
$$a^i_j = \sigma\left(w^i_j a^{i-1}_j + b^i_j\right) \tag{2.17}$$
$$a^{i-1}_j = \sigma\left(w^{i-1}_j a^{i-2}_j + b^{i-1}_j\right) \tag{2.18}$$

With the chain rule, it is easy to apply the same method to calculate how one weight in the $i-1$:th layer affect the output in the i:th layer in Equation 2.19.

$$\frac{\partial a^i_j}{\partial w^{i-1}_{j,k}} = \frac{\partial a^i_j}{\partial z^i_j}\frac{\partial z^i_j}{\partial a^{i-1}_j}\frac{\partial a^{i-1}_j}{\partial z^{i-1}_j}\frac{\partial z^{i-1}_j}{\partial w^{i-1}_{j,k}} \tag{2.19}$$

As the reader can observe this can be applied to see how one weight in the first layer can affect the output of the last layer as in Equation 2.20.

$$\frac{\partial a^L_j}{\partial w^1_{j,k}} = \frac{\partial a^L_j}{\partial z^L_j}\frac{\partial z^L_j}{\partial a^{L-1}_j}\cdots\frac{\partial a^1_j}{\partial z^1_j}\frac{\partial z^1_j}{\partial w^1_{j,k}} \tag{2.20}$$

As described in subsection 2.3.2, a loss function can be applied to the output of a network to get the loss of a prediction. In order to assess how the loss is affected by a weight, one can apply Equation 2.21.

$$\frac{\partial Loss_j}{\partial w^i_{j,k}} = \frac{\partial Loss_j}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j}\frac{\partial z^L_j}{\partial a^{L-1}_j}\cdots\frac{\partial a^i_j}{\partial z^i_j}\frac{\partial z^i_j}{\partial w^i_{j,k}} \tag{2.21}$$

The goal of doing backpropagation is to calculate how the weights and biases affect the loss function. The same method can be applied to calculate how one bias affects the loss as in Equation 2.22.

$$\frac{\partial Loss_j}{\partial b^i_j} = \frac{\partial Loss_j}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j}\frac{\partial z^L_j}{\partial a^{L-1}_j}\cdots\frac{\partial a^i_j}{\partial z^i_j}\frac{\partial z^i_j}{\partial b^i_j} \tag{2.22}$$

It is important to explain how partial derivatives can be calculated using vectors and matrices. By calculating how the first layer "A" affects the last layer "D" is described in Figure 2.17.

$\nabla_A D$ can be calculated by using Equation 2.23, Equation 2.24, and Equation 2.25. One can easily observe that $\nabla_A C = \frac{\partial B}{\partial A}\nabla_B C$. Furthermore, $\nabla_A D$ can be calculated with $\frac{\partial B}{\partial A}\frac{\partial C}{\partial B}\nabla_C D$, and this can trivially be calculated with a computer.

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \qquad B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} \qquad C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \qquad D = \begin{bmatrix} D_1 \end{bmatrix} \tag{2.23}$$

$$\nabla_A B = \begin{bmatrix} \frac{\partial B}{\partial A_1} \\ \frac{\partial B}{\partial A_2} \\ \frac{\partial B}{\partial A_3} \end{bmatrix} \qquad \nabla_B C = \begin{bmatrix} \frac{\partial C}{\partial B_1} \\ \frac{\partial C}{\partial B_2} \\ \frac{\partial C}{\partial B_3} \end{bmatrix} \qquad \nabla_C D = \begin{bmatrix} \frac{\partial D}{\partial C_1} \\ \frac{\partial D}{\partial C_2} \\ \frac{\partial D}{\partial C_3} \end{bmatrix} \tag{2.24}$$

$$\frac{\partial B}{\partial A} = \begin{bmatrix} \frac{\partial B_1}{\partial A_1} & \frac{\partial B_2}{\partial A_1} & \frac{\partial B_3}{\partial A_1} \\ \frac{\partial B_1}{\partial A_2} & \frac{\partial B_2}{\partial A_2} & \frac{\partial B_3}{\partial A_2} \\ \frac{\partial B_1}{\partial A_3} & \frac{\partial B_2}{\partial A_3} & \frac{\partial B_3}{\partial A_3} \end{bmatrix} \qquad \frac{\partial C}{\partial B} = \begin{bmatrix} \frac{\partial C_1}{\partial B_1} & \frac{\partial C_2}{\partial B_1} & \frac{\partial C_3}{\partial B_1} \\ \frac{\partial C_1}{\partial B_2} & \frac{\partial C_2}{\partial B_2} & \frac{\partial C_3}{\partial B_2} \\ \frac{\partial C_1}{\partial B_3} & \frac{\partial C_2}{\partial B_3} & \frac{\partial C_3}{\partial B_3} \end{bmatrix} \qquad \frac{\partial D}{\partial C} = \begin{bmatrix} \frac{\partial D_1}{\partial C_1} \\ \frac{\partial D_1}{\partial C_2} \\ \frac{\partial D_1}{\partial C_3} \end{bmatrix} \tag{2.25}$$

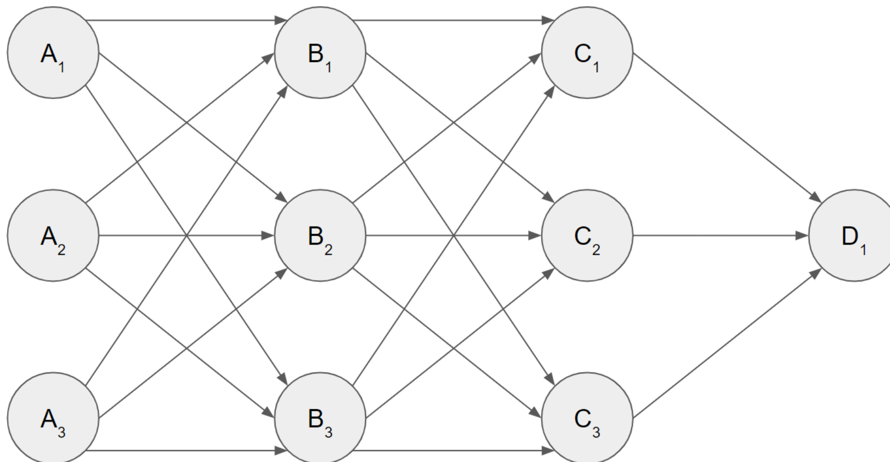

Figure 2.17: Example of a NN, one can see how the first layer A affects the last layer D.

With all this knowledge it is easy to compute how each weight and bias affects a given loss function. This is used by the optimiser to perform improvements in our model as described in subsection 2.3.4.

2.3.4 Optimiser

An optimiser is a bridge between the backpropagation and tuning of the parameters in the NN. The optimiser tweaks the weights and biases in the most optimal way for the network to make better predictions. The most basic optimiser is the gradient descent method as described in Equation 2.26. Some advantages of gradient descent are that it is easy to implement and is easy to understand. Some major disadvantages are that it may get trapped at a local minimum and it requires a lot of memory to calculate the gradient for a whole dataset.

$$\theta_{k+1} = \theta_k - \alpha \nabla J\left(\theta_k\right) \tag{2.26}$$

In the ML community the “go-to” optimisers, which also happens to be the most efficient optimisers, is the Adam [35] and AdamW optimiser. AdamW is a variant of the original Adam optimiser with added L2 regularisation and weight decay [36].

2.3.5 Overfitting

Overfitting is the phenomenon when the NN model stops generalising and starts to adapt solely to decrease the loss over the training set. There are several ways to reduce the effects of overfitting such as batch normalisation, dropout and early stopping which is explained in this section.

Batch Normalisation

Batch normalisation [37] sets the mean and variance of the input in each layer. As the name batch normalisation suggests the normalisation is done batch by batch. The mean is calculated as in Equation 2.27, the variance as in Equation 2.28, and the final output from the batch normalisation is described in Equation 2.29 for the i:th output in the k:th dimension, which would be 3 in the RGB image case.

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{2.27}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{2.28}$$

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}} \tag{2.29}$$

Dropout

Dropout is a stochastic method used during training by disregarding the output of some neurons in a specific layer. It means that during a forward pass (running

data through the network) and when doing backpropagation, the neurons that are affected by the dropout are ignored. This reduces the effects of overfitting and dropout layers are often placed in-between FCLs [38].

Early stopping

Early stopping is a method which chooses the version of a NN that performed the best during a training session, and when overfitting occurs to stop the training process. This is vital in getting a more successful NN, as overfitting often occurs eventually when training a NN. In Figure 2.18 the validation loss has the lowest value after the sixth epoch, the NN is overfitting during the seventh and eighth epoch. In most cases, the performance of the network will not improve, and early stopping is the best course of action.

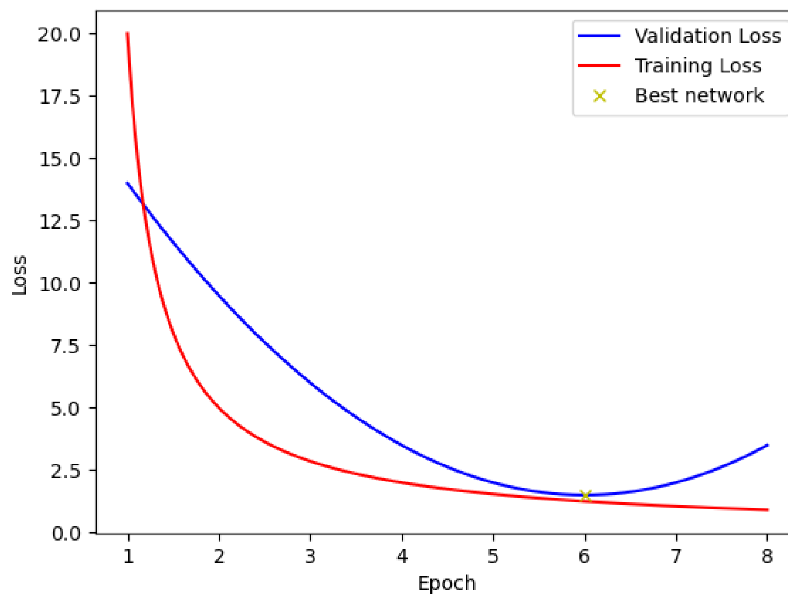

Figure 2.18: Example of early stopping.

3 Method

In this chapter, the method used in this thesis is described. In the first section of the method the development environment, both hardware and software are presented. The second section describes the dataset that has been used and how they have been processed. The third and fourth section of this chapter describes the custom kernels and the networks created for translation, rotation and scaling separately, also a baseline CNN will be presented. Thereafter the methods and choices related to training are explained.

3.1 Development environment

The computer which has been used for the training was equipped with a GeForce RTX 2080 Ti, with 4352 CUDA cores and 11 GB of GDDR6 memory, an Intel i7-6950X at a base frequency of 3.0 GHz and 128GB of DDR4 CL16 memory at 2.4 GHz. The code has been written in Python using the PyCharm editor. The PyTorch [39] machine learning library was used for developing and training the NN.

3.2 Dataset

This thesis aims to try alternative ways to detect rotation, translation and scaling between images. There is no apparent reason to perform tests on data related to the localisation. Instead, the data needed was images, with enough detail, that were pairwise, rotated, translated or scaled to each other. Two well-known data sets have been used that were then processed to create data sets for each of the regression tasks. MNIST [40] was the first dataset that was used, it consists of 60′000 training and 10′000 testing images that depict handwritten digits, the images are grayscale and have a size of 28x28 pixels. This dataset enabled fast testing, however, due to the low resolution, it was quickly concluded that we needed new data with higher resolution. Throughout the rest of the work, efficient testing of new ideas was preferred. The dataset STL-10 [41] [42] was chosen, it has higher resolution, but still low enough to make testing efficient. The STL-10 dataset was originally designed for image classification, it contains 100′000 unlabelled images, 5000 labelled training images, 8000 testing images. All images are RGB images and contain vehicles and animals with a resolution of 96x96. The unlabelled images were used for training, the labelled training images for validation and the labelled testing images for testing.

Figure 3.1: Example images from the MNIST dataset. Consisting of images with resolution 28x28, depicting handwritten digits in grayscale.

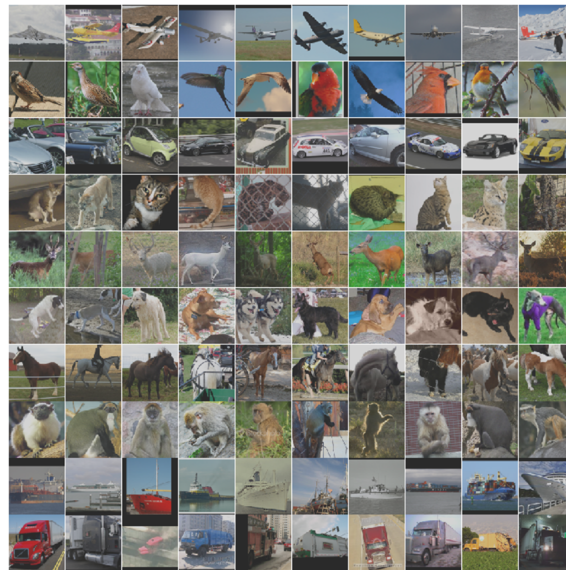

Figure 3.2: Examples of images from the STL-10 dataset. Consisting of colour images with resolution 96x96.

To fit the images for the translation task, each image is copied and translated. To not always draw one picture from the centre of the image, the first image centre is first drawn at a random position, but not too close to the borders. Then the next image is uniformly translated ±10 pixels along the horizontal and vertical axis, from the first image. The target vector is the number of translated pixels in each direction. The reference image and the translated image are stacked on top of each other. To remove any black border from the translated image, both images are cropped to a size of 48x48 pixels, this gives an input of shape [6, 48, 48] for the STL10 dataset. Some examples from the dataset created for translation are shown in Figure 3.3.

Figure 3.3: 10 example image pairs from the constructed translation dataset.

To fit the images for the rotation task, each image is copied and rotated. The first image is rotated uniformly from 0° to 360°. Then the rotation of the second image is drawn uniformly from a distribution of ±40° to the first image. All rotations are around the centre of the image, an affine transformation with bicubic resampling, and the target vector is the number of degrees between the two images. The reference image and the rotated image are later stacked on top of each other. Examples from the dataset with rotated images are shown in Figure 3.4. To remove any black border from the rotated image, both images are cropped to a size of 48x48 pixels, this gives an input of shape [6, 48, 48] for the STL10 dataset.

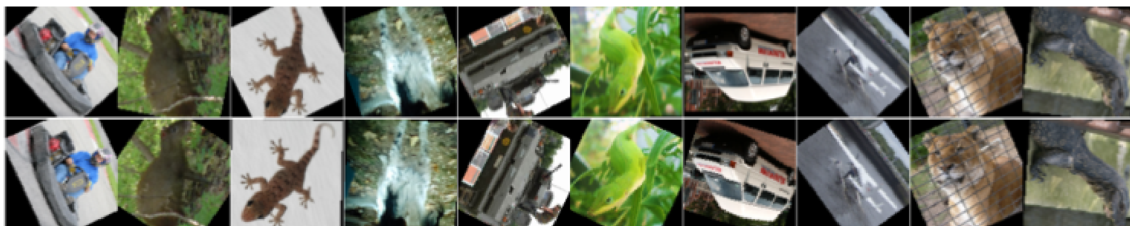

Figure 3.4: 10 example image pairs from the constructed rotation dataset (before being cropped to 48x48).

To fit the images for the scaling task the first image is scaled down to 0.5 of the original size and then centre-cropped to 48x48 pixels. The second image is a copy of the same original image scaled uniformly between 0.5 and 1 and then centre-cropped the same way as the first image. Then the first image depicts the whole original image but is half the size, the second image is also half the size but is a zoomed-in version that does not depict the whole original image. The target vector is the scaling factor from 1 to 2 between the two new images, meaning the scale of the second image relative to the first image. The reference image and the scaled image are later stacked on top of each other. The input shape is the same as in the translation and rotation datasets, [6, 48, 48] for the STL10 dataset. Examples from the dataset with scaled images are shown in Figure 3.5.

Figure 3.5: 10 example image pairs from the constructed scaling dataset.

3.3 Kernel shapes

This section explains how the input kernel shape and stride are designed to optimise for predicting translation, rotation and scaling. The example input image can be found in Figure 2.1, and an example of how a square kernel would process the image is depicted in Figure 3.6. The difficulty in backpropagation does not change with the kernel shape, as it only changes the receptive field of the kernel.

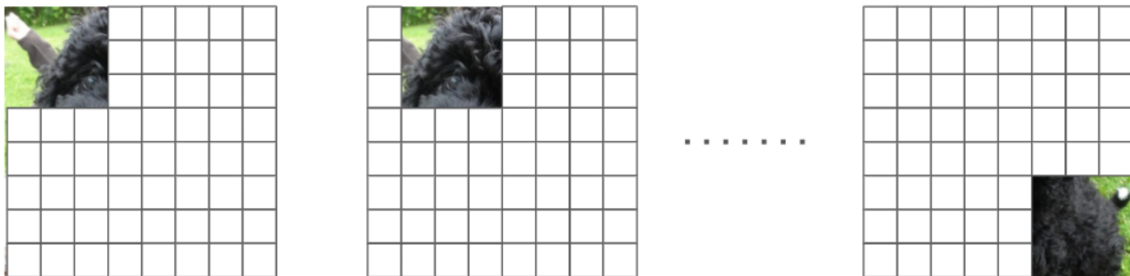

Figure 3.6: Example of how a 3x3 square kernel stride over an input image. Note that the actual pixel size of the image is much smaller than the white example pixels.

3.3.1 Translation

When trying to detect the amount of translation in an image, is there a better way to do this than using a normal square kernel CNN? A hypothesis is that a rectangular kernel that is very thin in one direction but covers the whole image in the other direction, could be better at detecting translations along one dimension. A network with this kind of layer will be referred to as a *translating-convolutional neural network* (T-CNN). A T-CNN can be used twice (with shared weights), for example, if the network is trained to detect translation along the horizontal axis, the image can be rotated 90°, which can vastly reduce the training time and size of the network. This example can be found in Figure 3.7.

Implementation

The implementation of rectangular kernels was done with the built-in Conv2D function in PyTorch. The input batches were permuted such that the width of the image instead was considered the channel depth. The kernel size was set to (3,1) and the stride to (3,1). The two input images were run through the network twice, first

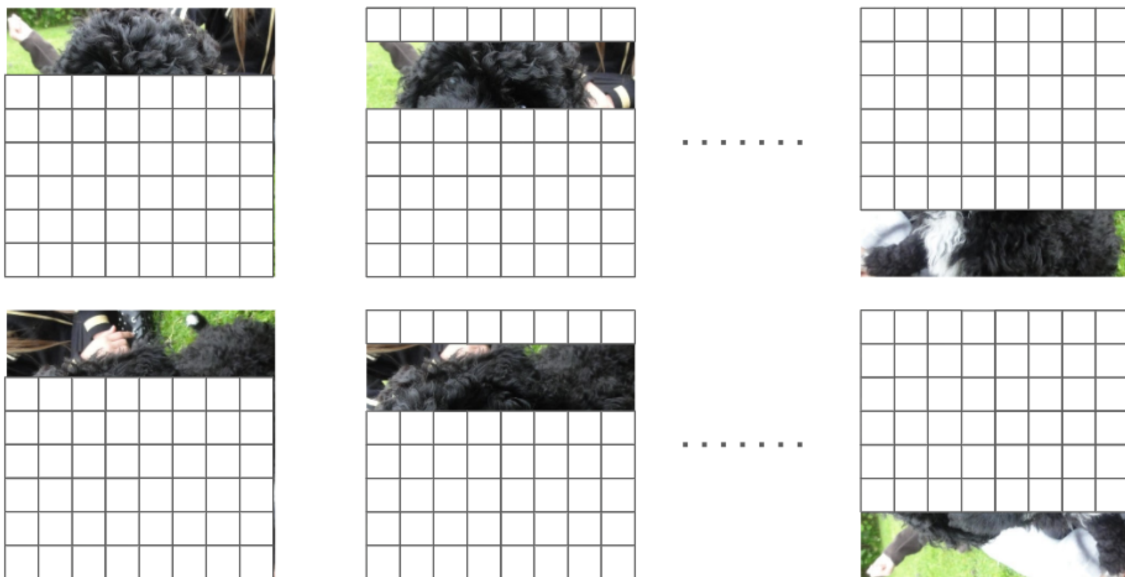

Figure 3.7: Example of how the first layer of the T-CNN processes the input image. Note that the actual pixel size of the image is much smaller than the white example pixels.

with the original images and then with the images rotated 90°. With this setup, the network predicts a vertical translation with the original images and a horizontal translation with the rotated images. See Figure 3.8 and Figure 3.9 for a visual explanation.

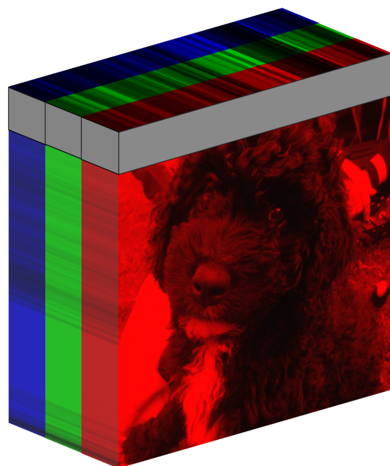

Figure 3.8: Input for regression along the vertical axis.

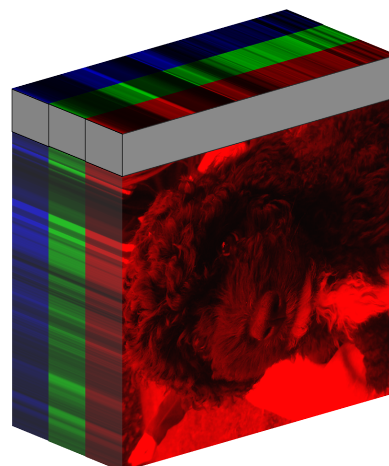

Figure 3.9: Input for regression along the horizontal axis.

3.3.2 Rotation

Our hypothesis of how a kernel can improve detecting rotation is by the shape of a circle sector. The kernel strides around the centre of the image and not along the horizontal and vertical axes. In this way, the NN can detect the same features

in the image even though the target image is rotated. Thus the feature will only be shifted in the output. A network with this kind of layer will be referred to as a *rotating-convolutional neural network* (R-CNN). The parameters that need to be set for this layer are the output size, the number of pixels in each circle sector and, how many steps the filter takes around the image. An example of this kind of kernel can be found in Figure 3.10.

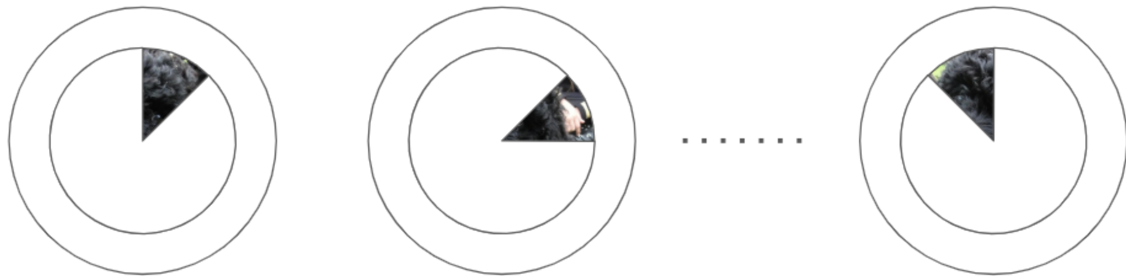

Figure 3.10: Example of how the first layer of the R-CNN processes the input image. Note that the actual width of the kernel is smaller than in the example.

Implementation

While the visualisation of the circular convolution layer is straight forward the task itself is more difficult in practice. The shape and size of the kernel need to be conserved when striding around the image. The image consists of a bitmap structure (as described in section 2.1 and a circle sector can be approximated to a bitmap. However, a problem arises when the circle sector is rotated an arbitrary amount and needs to find the corresponding one to one matches in the new area. The output of a kernel needs to be interpreted the same way regardless of the current angle of the filter. The implementation for the circular convolution is based upon nearest neighbour resampling when rotating bitmap images. The implementation conserves size and shape, however, it suffers from some sparsity, as can be seen in Figure 3.11.

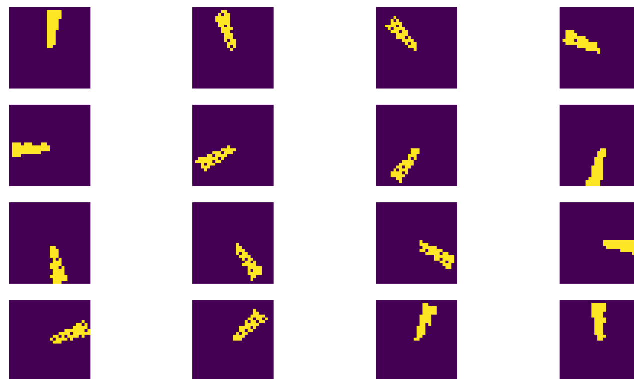

Figure 3.11: Example of how a kernel strides over an image. Image size (28x28), with a kernel of 48 pixels, rotated around the image in 16 steps.

3.3.3 Scaling

As previously stated, a question was asked whether there is a better way to detect scaling than using a normal CNN. An idea of using “frame-shaped” kernels with a variety of sizes are used to analyse the images. An example of this kind of kernel can be found in Figure 3.12. A network with this kind of layer will be referred to as a *scaling-convolutional neural network* (S-CNN).

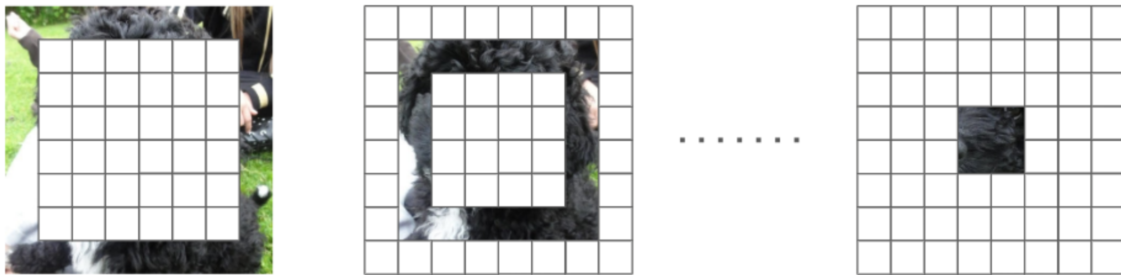

Figure 3.12: Example of how the S-CNNs first layer process the input image. Note that the actual pixel size of the image is much smaller than the white example pixels (as for all following examples).

Implementation

The implementation of the scaling layer has much in common with the FCL, as the kernel does not stride their fixed position. Instead of having, for example, eight filters that stride over the image, there would be eight filters for every frame size, currently each frame has a width of one pixel. The total number of outputs from the scaling layer would then be eight times the number of frames. An image as in Figure 3.12 with 8x8 pixels would yield four different frame sizes (if the frame width is one and “stride” one).

3.4 Networks

The baseline CNN in Figure 3.13 has the same structure as a “classic” classification network. It was trained with two outputs for the translation test and one output for the rotation and scaling test respectively. The structure for the translation, rotation and scaling networks with the custom input layer can be found in Figure 3.14, Figure 3.15 and Figure 3.16. To be able to compare the networks with custom input layer to the baseline CNNs in a fair way all networks were designed with the same number of layers. Each network has six convolution and three fully connected layers. The networks are designed such that the number of neurons in each layer is similar.

3.4.1 Baseline CNN

The baseline CNN is visualised in Figure 3.13.

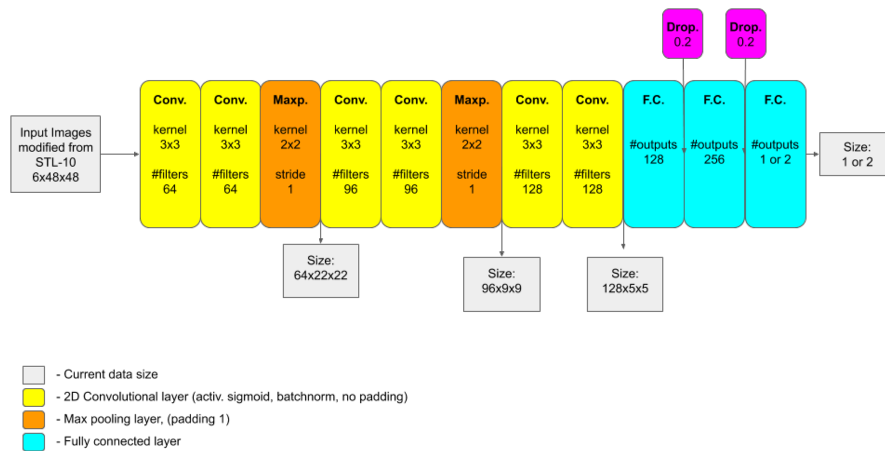

Figure 3.13: Visualisation of the baseline CNN network.

3.4.2 Translation network (T-CNN)

The T-CNN is presented in Figure 3.14. The first layer has the wide custom kernel, then follow five convolution layers and then three fully connected layers.

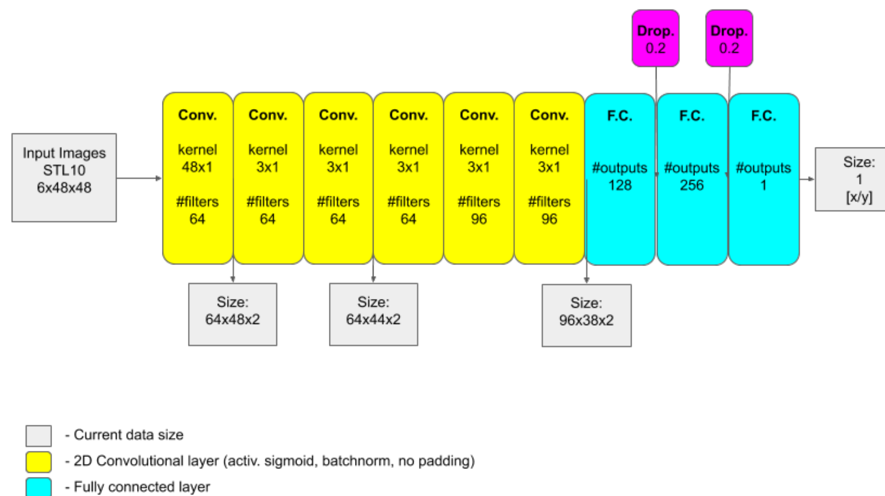

Figure 3.14: Visualisation of the T-CNN network.

3.4.3 Rotation network (R-CNN)

The R-CNN is visualised in Figure 3.15. The first layer in this network is the circular convolution layer. The filter size was chosen to 128 pixels. The filters were rotated around the image in 50 steps and 512 filters were used, followed by five convolution layers with a kernel of 3x1. This means that the results from three circle sectors were grouped together and that the two images were treated separately. In the last three fully connected layers the data from both images processed by the previous layers are finally processed together.

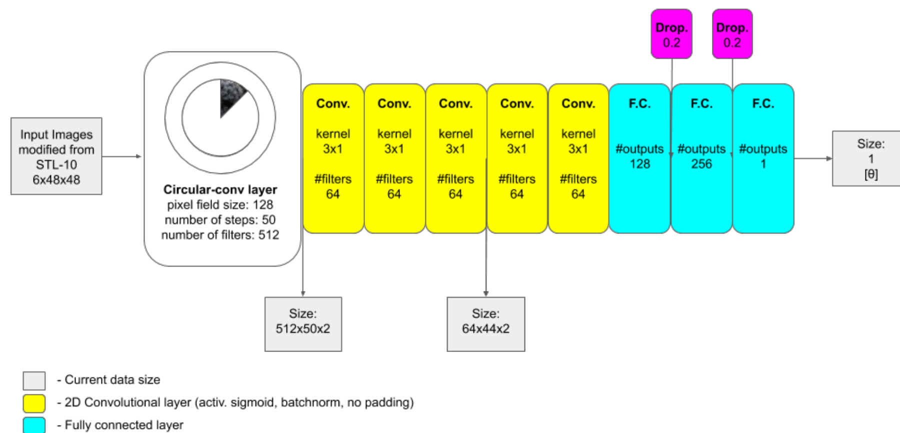

Figure 3.15: Visualisation of the R-CNN network.

3.4.4 Scaling network (S-CNN)

The S-CNN is visualised in Figure 3.16 and consists of the scaling layer, five convolution layers and three fully connected layers. In the first convolution layer, neurons from three different frame sizes are weighed together.

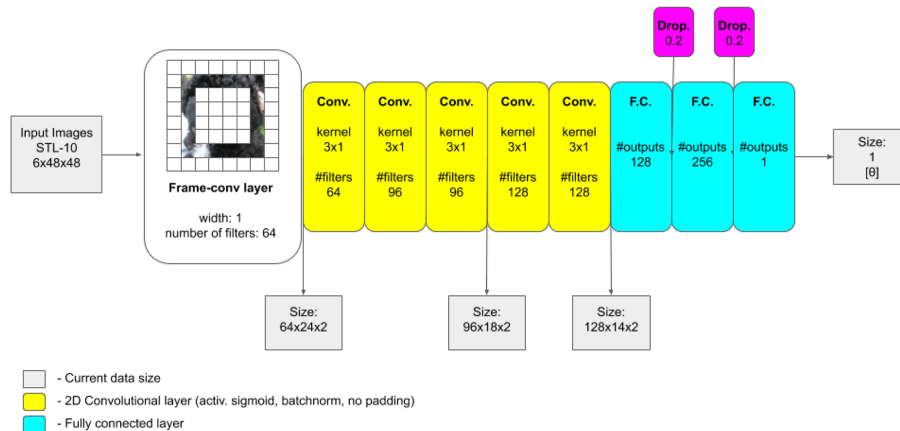

Figure 3.16: Visualisation of the S-CNN network.

3.4.5 Training

For the training of the NNs, the AdamW optimiser from section 2.3.4 was used with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and weight decay to 10^{-2}. The chosen loss function was the MSE loss as described in subsection 2.3.2. The hyperparameters were manually tuned for each network. The batch size and learning rate gave the best results at the same values for all NNs. The batch size was set to 2^{10}. The initial learning rate was set to 0.005. If the validation loss increased from one epoch to another, the learning rate was decreased by multiplying the current learning rate by

0.7. In this way, the learning rate was able to decrease to 10^{-8}. The NN is trained until the validation loss of the epoch had not improved since ten epochs.

3.4.6 General design

The goal of this thesis is to see whether a CNN with a new input layer can be more successful in regression tasks than a classification CNN. To be able to answer this question we need to constrain the complexity of the layers to be able to make fair comparisons. This meant that we chose the total number of trainable parameters in the network to roughly be the same. The numbers of hidden layers with trainable parameters should also be the same in both networks. As described in subsection 2.3.2 the weights and biases were tuned during training to minimise the MSE loss. However, comparing the loss does not give any accessible information of the NNs performance except that a lower loss is generally better. In the result section, we present histograms of the errors in the network's predictions. With interesting values such as bounds on the distribution, mean, median and max error.

4
Results

The NNs performance was evaluated using 8000 image pairs were translation, rotation or scaling were estimated separately. To assess how good a specific estimation was, the error was chosen to be the difference between the true value and the estimated value. The goal was to get an error distribution with as many errors as possible to be close to zero. The metrics that were chosen to focus on, when evaluating the NN, were the mean, median and max absolute value of the errors over the whole test set. Histograms of the estimation errors are provided in Figure 4.1 to 4.6 and can be used to get a visual representation of the network's performance. It is important to note that the vertical axis is set to a logarithmic scale. To make readouts of the distribution easier, the histograms are marked with 80%, 90% and 95% bounds, in a similar fashion as Gaussian distributions are often marked with 68%, 95% and 99.7%. The initial distribution is shown in red, which can also be interpreted as the error if a network would always guess the mean of the true values of the dataset. For example, if the translation is between ±10, the red error distribution always corresponds to estimating that the translation between the two images is zero.

4.1 Translation

As stated in section 3.2, the translation dataset consists of discrete uniform distribution with bounds of ±10 pixels translated. The copied image is translated along both the horizontal and vertical axis. The T-CNN was trained for 40 epochs and the baseline CNN was trained for 51 epochs, due to early stopping. The figure of the estimated translation is in Figure 4.1 and Figure 4.2. Additional results can be found in Table 4.1. The results are discussed further in subsection 5.2.1.

	T-CNN	**Baseline CNN**
mean |error|	0.75	0.26
median |error|	0.49	0.19
max |error|	15.23	8.39
validation loss	0.01293	0.00215
95% bound	[−2.66, 2.46]	[−0.78, 0.7]
90% bound	[−1.81, 1.71]	[−0.58, 0.53]
80% bound	[−1.21, 1.12]	[−0.42, 0.37]

Table 4.1: Results from the translation tests. The values in the table is for translations along both directions.

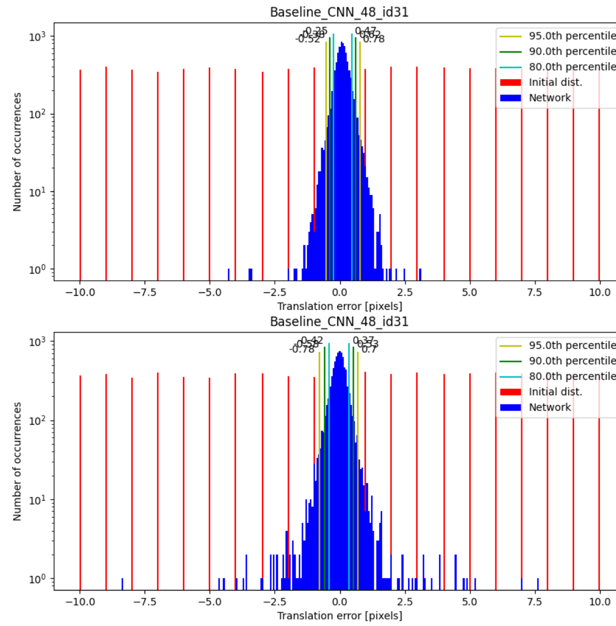

Figure 4.1: Translation Baseline CNN histograms. One histogram for each translation direction.

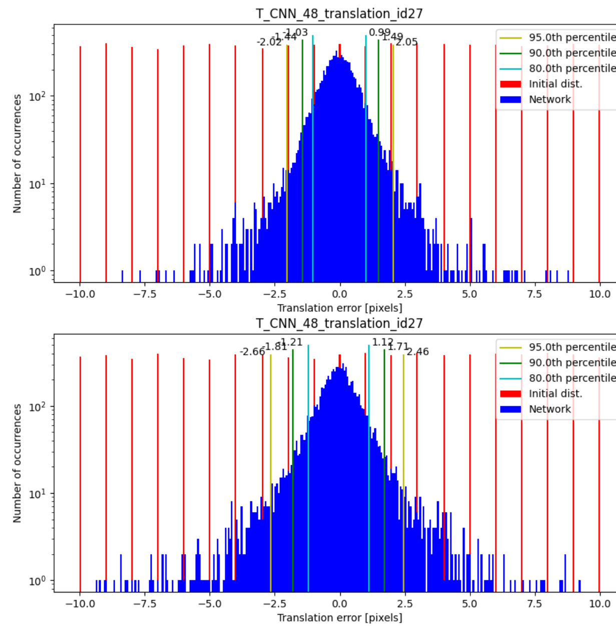

Figure 4.2: Translation T-CNN histograms. One histogram for each translation direction.

4.2 Rotation

As described in section 3.2 the images are uniformly rotated using an affine transform with bicubic resampling and the same datasets was used for both networks. The images were rotated so that the relative rotation of the image pairs was uniformly distributed between ±40°. The R-CNN was trained for 37 epochs and the baseline CNN was trained for 57 epochs, due to early stopping. Histograms of the rotation results can be found in Figure 4.3 and Figure 4.4. Some interesting values can be found in Table 4.2. Overall the baseline CNN performed better than the R-CNN, the results are further discussed in subsection 5.2.2.

	R-CNN	**Baseline CNN**
mean |error|	0.62	0.39
median |error|	0.47	0.27
max |error|	7.24	24.64
validation loss	0.00042	0.00022
95% bound	[−1.66, 1.90]	[−1.11, 1.14]
90% bound	[−1.20, 1.31]	[−0.81, 0.87]
80% bound	[−0.86, 0.95]	[−0.52, 0.61]

Table 4.2: Results from the rotation tests

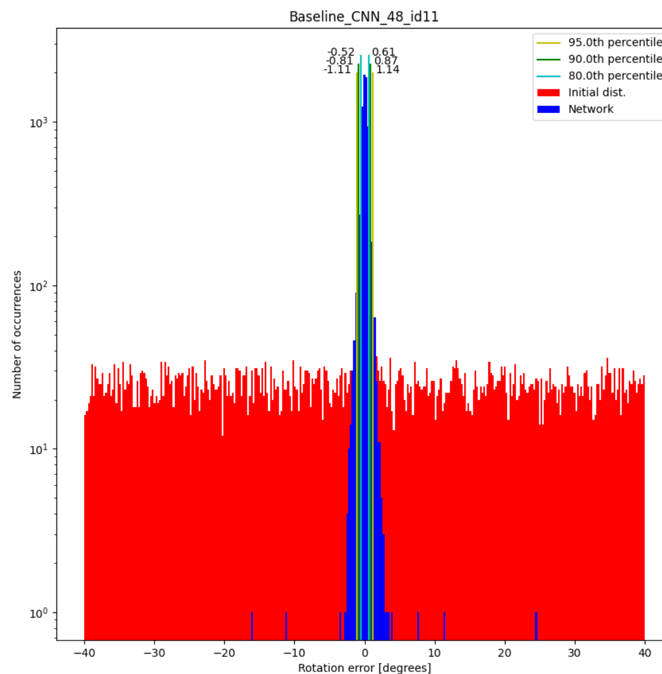

Figure 4.3: Rotation CNN histogram.

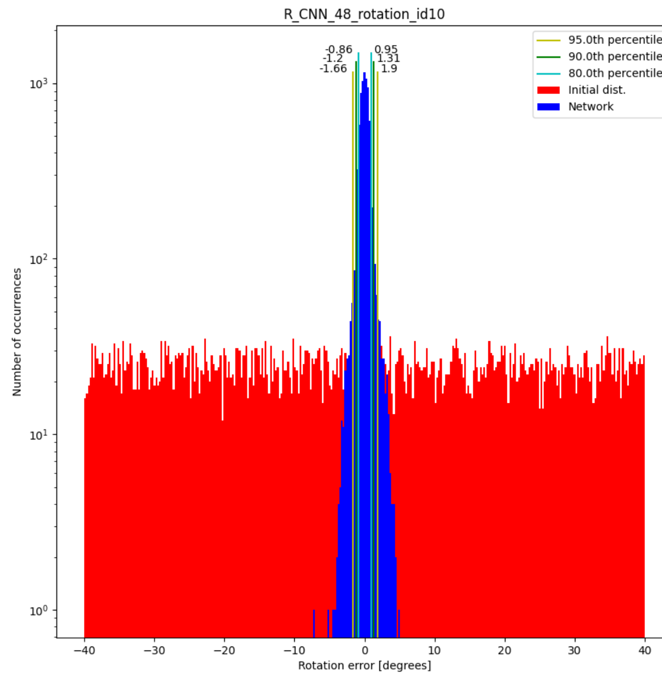

Figure 4.4: Rotation R-CNN histogram.

4.3 Scaling

The scaling was done by resizing and cropping the image to the correct size of 48x48 pixels. Histograms of the scaling results can be found in Figure 4.5 and Figure 4.6. The horizontal axis is from -0.5 to 0.5 and the error is the difference between the estimated scaling factor and the true scaling factor. Additional results can be found in Table 4.3. The S-CNN was trained for 45 and the baseline CNN was trained for 59 epochs, due to early stopping. The results are discussed in subsection 5.2.3.

	S-CNN	**Baseline CNN**
mean |error|	0.02	0.01
median |error|	0.02	0.01
max |error|	0.44	0.12
validation loss	0.00297	0.00087
95% bound	[−0.07, 0.05]	[−0.04, 0.03]
90% bound	[−0.05, 0.04]	[−0.03, 0.02]
80% bound	[−0.04, 0.03]	[−0.02, 0.01]

Table 4.3: Results from the scaling tests

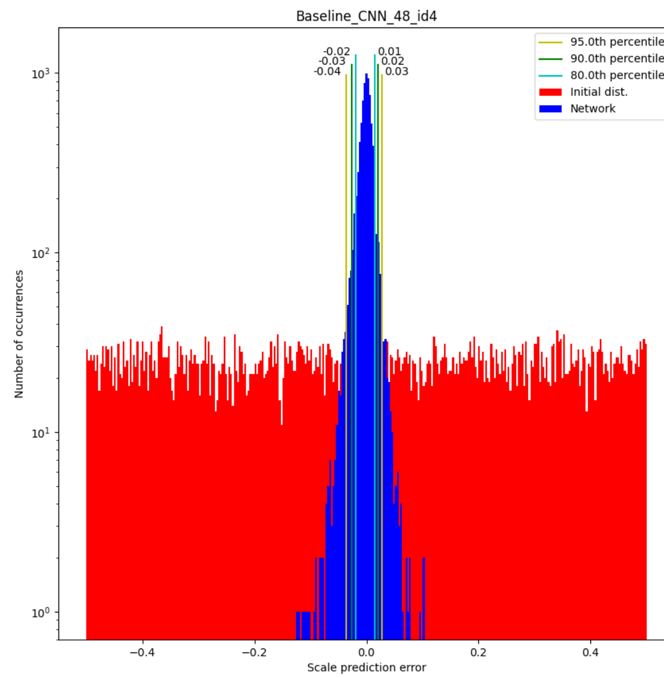

Figure 4.5: Scaling CNN histogram.

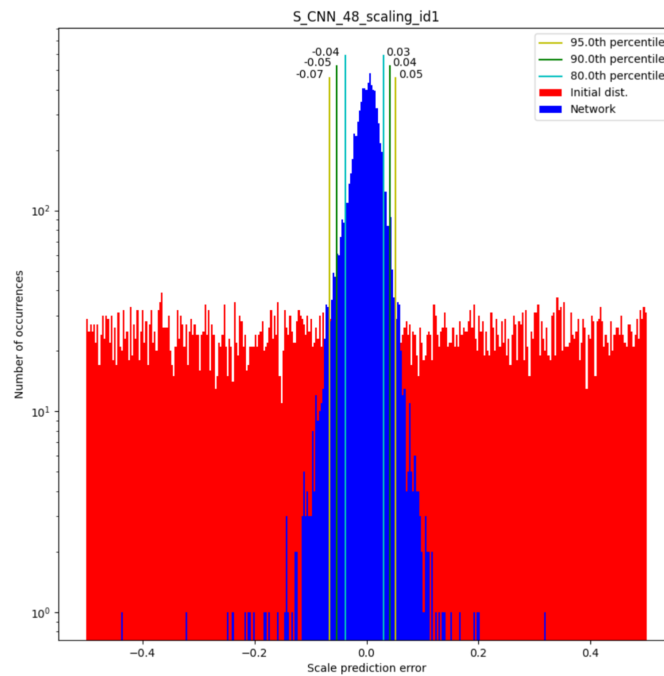

Figure 4.6: Scaling S-CNN histogram.

5 Discussion

In this chapter, the results from the previous section, and general thoughts of what worked well and what could have been made differently are discussed.

5.1 All networks converge

When training a NN, the convergence of the validation loss is vital. We can conclude that all networks with the custom layers succeeded in the convergence aspect. Convergence during training was obtained in all cases. This is a success and proves that the networks designed can be used for the task at hand. The convergence of the training loss does not, however, give any guarantee that the performance is comparable to other methods or sufficient for actual use. In the histograms in Figure 4.1 - 4.6 the initial distribution shows how the test set is distributed. We can see that all networks have a smaller spread than the initial distribution. Therefore we can conclude that the designed networks perform at a much higher level than what a network that always guesses the mean of the true values of the dataset would.

5.2 Comments on results

The results for each task will be discussed separately in the sections below.

5.2.1 Translation

We can observe that the T-CNN does not outperform the baseline CNN. A reason behind this could be that the custom layer is more sensitive to translation along two dimensions at the same time. The baseline CNN is estimating the translation along both the x and y dimensions at the same time, and can therefore also learn to use the knowledge of both estimations to decrease the effect of the “noise” caused by translation along the other dimension. The T-CNN makes two separate predictions, first along the horizontal axis and then along the vertical axis. We can understand that the T-CNN is more sensitive to translation along with two directions by testing the performance of the networks when the image is only translated along one dimension. In section A.3 results from tests along only one dimension is presented. One can conclude that the difference in performance between the baseline CNN and the T-CNN is almost none when the translation was only along one dimension. It would have been interesting to test a network structure were two convolution

parts of the network worked separately but with shared weights and then were connected together in the last three fully connected layers, see Figure 5.1 for a visual explanation.

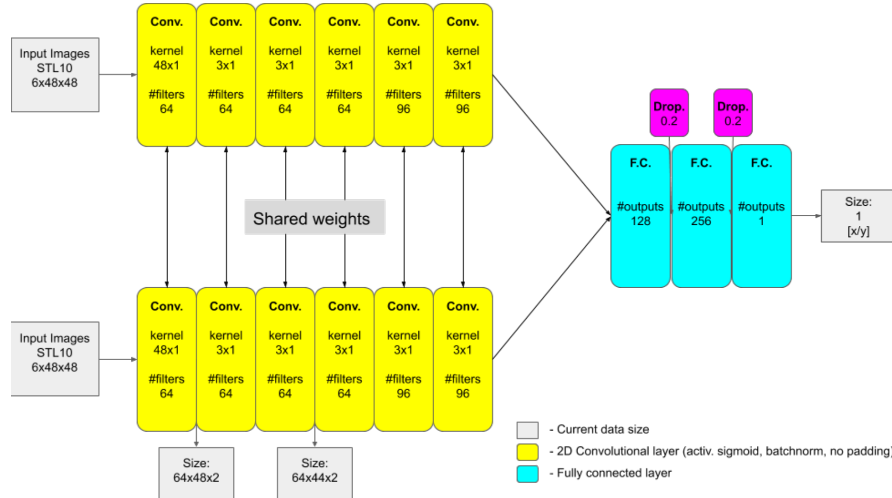

Figure 5.1: Example of an alternative, siamese T-CNN network.

5.2.2 Rotation

With the rotation estimation, we conclude that both networks perform very well on the test set. Both networks get a mean of the absolute error below 1° and a median of the absolute error below 0.5°. One can see that the performance of the baseline CNN is slightly better than the R-CNN in all but one aspect. The baseline CNN has more outliers and the outlier errors are worse. This is perhaps not as surprising when it comes to the intuition behind the two layers. The baseline CNN draws out features from the images, it uses these features to determine the degree of rotation between the images, when this then fails the error is quite large. In the R-CNN the intuition of the rotational layer was to stride along the rotational axis, one could argue that the precision of this method seems weaker, but with greater robustness when it comes to outliers. The robustness of the R-CNN gives hope that the intuition behind this layer-design is correct. Therefore, we think that concept of a rotation layer should be researched more. It is probable that a better implementation could outperform the baseline CNN and at the same time be more robust by having fewer outliers.

5.2.3 Scaling

The S-CNN network did not perform better than the baseline CNN, it converged but did not get as good accuracy. The intuition behind this scaling solution was not as strong as the R-CNN or T-CNN. There might be better ways to adapt a NN to estimate scaling between images. The biggest drawback with the S-CNN solution is that the weights are not well utilised. The notion of convolution in the scaling layer was not implemented, instead, the filters were fixed and only used on

a predefined area, i.e. several filters for each frame size. This led to a very large number of weights and inefficient usage of these weights in the NN. The current S-CNN implementation does not scale well with bigger inputs contrary to the baseline CNN. The reason that the filter was fixed was that we did not find a reasonable way to stride the filters since the frames change in size.

5.3 Image resolution

The initial testing was performed on the MNIST dataset. We noticed that the image resolution was constraining the NNs ability to make accurate predictions. Tests showed that rotations up to even $\pm 5°$ had a very small impact on the images. This resulted in a decision to change to the STL-10 dataset with higher resolution images. In the STL-10 dataset, small differences in rotation yielded bigger differences between the image pairs. Since the networks performed differently after the change of dataset, it was concluded that the resolution was no longer the main bottleneck for the performance. However, the resolution of the 48x48 images is still low compared to the images which would be useful in a real-world scenario. Therefore, it would be interesting to make these custom layers and their networks designed for bigger input images, possibly a suitable area for a future thesis.

5.4 Implementation

It requires a lot of thought when implementing a new layer. There are many design choices that need to be taken into account such as kernel shapes, size and how to stride over the input. The implementations also have to be efficient enough to be able to be used in thorough testing. It is clear that we have not been able to test everything. We still have new ideas and new variants that would be interesting to test, but there was no time for more work within this thesis. We will instead try to cover as much of this here in the discussion and the following chapter.

It could be that these new network layers we have designed, will not be able to outperform a CNN but that should not be concluded yet. There are much more that can be tested, changed, and tuned. We think that at least the rotational layer shows a great deal of potential. The translation layer could also be promising, but the difference between the T-CNN and the baseline CNN is not more than the width of the kernel, so even if this would be developed further we think that the T-CNN and the baseline CNN will perform in a similar manner. The S-CNN, in its current version, does not show much of potential and also, as previously mentioned, it is not scalable to bigger image inputs. Since we see the most potential in the rotational layer, it will be discussed further in the following section.

5.4.1 Rotation layer

As we commented on earlier in the discussion the R-CNN and the baseline CNN have similar performance. Even though the baseline CNN has a lower absolute

mean error and absolute median error, we can see that the baseline CNN has far more outliers and some quite large errors. The method for the rotation layer that was used to get the result is described in Figure 3.3.2 but earlier in the project, we considered a different implementation.

The disregarded implementation had a fixed shaped kernel and, instead of having the filter stride around the image, the input image was rotated with an affine transformation with a bicubic resampling filter. This implementation did not suffer from the sparsity circle sector problem. In section A.2, the disregarded implementation is briefly described and an example of early test results for the R-CNN on the 28x28 MNIST dataset is presented. Note that those results are not directly comparable with the other results in the report since both the networks were different and it was trained and tested on a different dataset. Examples from these 28x28 MNIST data sets are presented in section A.1. However, the first implementation of the R-CNN gave great results, which made us think that this kind of implementation would have worked better. A huge drawback was that it required all circle sectors from the images to be extracted before training. The pre-processing time took hours on the images MNIST dataset and therefore this method was discarded when we moved to the STL-10 dataset. We, therefore, think that it would be interesting to develop a new implementation, that does not need to pre-process the images but still manages to implement the filters in a better way. Exactly how this should be done is not clear. Why this is a challenge can be shown with an example; see Figure 5.2. The task to implementing these filters in such a way that the same amount of pixels is always withdrawn, at the correct positions, and with a consistent interpretation can be very difficult, but maybe a better solution can be found.

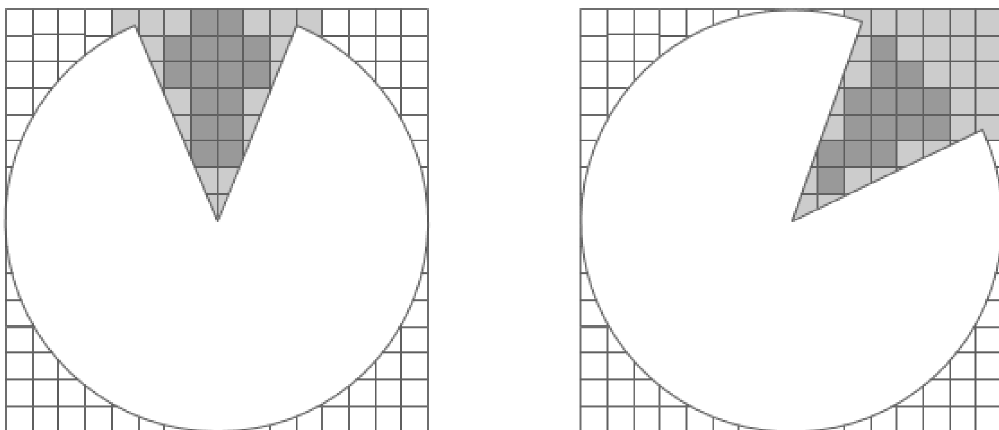

Figure 5.2: Images showing that the interpretation of the inputs to a circle sector filter at two different angles is not trivial.

5.4.2 Network layout

A potential drawback with the layers we have built is that the layers cannot be stacked upon each other. We have one custom layer and then the rest of the network has a normal CNN structure. We have also not been able to figure out a good way of pooling the data. It has been hard to know how to design the networks since they do not have any standard layout. The focus of our work has been to design the new layers, something that would need to be considered further in future research would be to put more effort in how the networks around these layers should be designed.

5.4.3 Network comparison

As described in subsection 3.4.6 the network design was constrained due to a couple of factors. One of the more difficult tasks in this thesis was to do fair comparisons of the networks. Maybe the fairest way would have been to not balancing the number of layers in the networks, number of neurons, or number of parameters. The balancing of the networks was done to make the comparison and tuning easier but it might have made the comparison more unfair instead of fair.

6
Conclusion

We have created three kinds of custom NN layers. It can be considered a success because the NNs with the custom layers converge and show potential. Even though the NNs with custom layers did not outperform the baseline CNNs. It was difficult to make comparisons because of the different network structures. Manual tuning of parameters makes it hard to know if an optimal or close to optimal tuning has been reached. We conclude that the idea of custom kernels and layers is an interesting research field.

Application towards localisation

As mentioned in the problem formulation in section 1.2. the motivation behind designing new layers is to do pose regression. The aim was to utilise it in the field of localisation and evaluate if the custom shapes for kernels and stride along the axis of interest could increase performance and robustness. The aim to see if a new NN can yield better results compared to "classical" CNN in the RPR task. We can conclude that we have been able to design these three new types of layers. Even though none of them currently seems better suited to solve the RPR task, when comparing to a baseline CNN with only square kernels. The area should be researched further before data-driven approaches such as CNNs can be able to surpass the performance of IR and compete with classical VO and V-SLAM methods.

6.1 Future work

In the T-CNN, we think that it might not be the best way to perform estimation of translation along the two dimensions separately. Instead, we think it would be more promising to let the T-CNN make both predictions in conjunction. In future work for the translation task, we suggest a new NN design were the estimation of translation is estimated along both dimensions at the same time or at least in a more cooperative way.

In the R-CNN we suggest a different implementation of the rotation layer to avoid pixel sparsity but still preserving shape and size. A new layer implementation must be considered thoroughly such that the filter weights are interpreted in the same way no matter from which angle the filter reads.

If the S-CNN should be used in future work, we believe that the filters must be treated differently. One could consider changing the frame shape to something more

suitable such that the filter can stride over the image. How this would work is currently not clear to us. Another direction would be to reconsider the problem of scaling estimation and see if a new intuitive understanding of the problem can be used to design a different kind of layer.

What would be a bigger step and even more interesting would be to find ways to combine all these three methods such that a network can make regression of similarity transformed images. Instead of using image inputs which are synthetically translated, rotated or scaled, it would be interested to use images from a simulator or from real scenarios, to estimate translation, rotation and scaling, from two consecutive images.

6.1.1 Relative pose regression

Custom networks designed to detect translation, rotation and scaling can be used in RPR scenarios. RPR is possible given an ACV with a camera pointed upward and forward in the direction of travel as in Figure 6.1. The upward-facing camera can use a rotation network to approximate if the ACV has turned and a translation network can perform estimation of general movement. The forward-facing camera can use a scaling network to estimate movement forward going and a translation network can estimate turns and if the car moves up or down on a hill. This could be done with both real-world data, but also tested with simulated data, such as from the open-source simulator Carla.

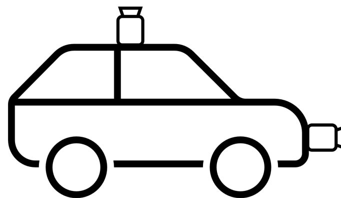

Figure 6.1: Example of a car with an upward and a forward facing camera.

6.1.2 Absolute pose regression

Although somewhat farfetched, this research could be used to perform APR. Instead of making estimations between two equal but translated, rotated, and scaled images one could use an image and a map cut-out. This was done in the Linderoth and Lundqvist thesis [27], were a 2D LiDAR scan was remade to an image and then compared to a cut-out from a top view 2D LiDAR map. The goal could then be to develop the ideas further such that the images can be compared to bigger map parts and finally to a full map and then what we have would be APR, finding where in the map the image fits in.

Bibliography

[1] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

[3] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, 2011.

[4] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[6] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[7] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[8] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[9] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[10] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[11] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[12] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[13] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[14] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 134–144, 2019.

[15] A. Torii, J. Sivic, and T. Pajdla. Visual localization by linear combination of image descriptors. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 102–109, 2011.

[16] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[17] Amir Roshan Zamir and Mubarak Shah. Accurate image localization based on google maps street view. In *European Conference on Computer Vision*, pages 255–268. Springer, 2010.

[18] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla. 24/7 place recognition by view synthesis. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1808–1817, 2015.

[19] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[21] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Visual localisation demo.

[22] Noha Radwan, Abhinav Valada, and Wolfram Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018.

[23] Emilio Parisotto, Devendra Singh Chaplot, Jian Zhang, and Ruslan Salakhutdinov. Global pose estimation with an attention-based recurrent network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 237–246, 2018.

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[26] Simon Bastås and Robert Brenick. Outdoor global pose estimation from rgb and 3d data. Master's thesis, 2019.

[27] Sabina Linderoth and Annika Lundqvist. Map-based localization using lidar and deep neural networks. Master's thesis, Chalmers University of Technology, 2019.

[28] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixé. Understanding the limitations of cnn-based absolute camera pose regression. *CoRR*, abs/1903.07504, 2019.

[29] Qunjie Zhou, Torsten Sattler, Marc Pollefeys, and Laura Leal-Taixe. To learn or not to learn: Visual localization from essential matrices. *arXiv preprint arXiv:1908.01293*, 2019.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] 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.

[34] Soumith Chintala Gregory Chanan Adam Paszke, Sam Gross. Pytorch website. `https://pytorch.org/docs/stable/nn.html`. (accessed: 21.04.2020).

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[39] Soumith Chintala Gregory Chanan Adam Paszke, Sam Gross. Pytorch website. `https://pytorch.org/`. (accessed: 21.04.2020).

[40] Christopher J.C. Burges Yann LeCun, Corinna Cortes. Minst database website. `http://yann.lecun.com/exdb/mnist/`. (accessed: 21.04.2020).

[41] Andrew Y. Ng Adam Coates, Honglak Lee. Stl10 database website. `http://cs.stanford.edu/~acoates/stl10`. (accessed: 21.04.2020).

[42] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

A
Appendix

A.1 Examples from the first datasets

Below are some example image pairs from the first datasets created from the MNIST dataset. The initial images had a 28x28 pixel resolution.

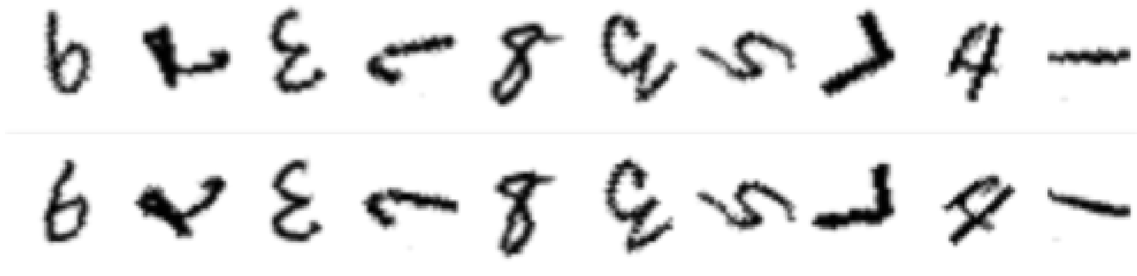

Figure A.1: 10 example image pairs from the constructed rotation dataset.

Figure A.2: 10 example image pairs from the constructed translation dataset.

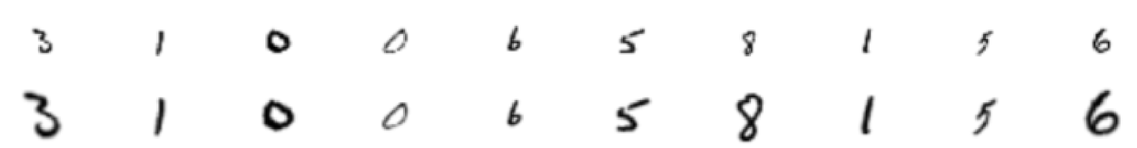

Figure A.3: 10 example image pairs from the constructed scaling dataset.

A.2 First testing of R-CNN and baseline CNN on 28x28 images

The results below from a baseline CNN and our R-CNN but with a different implementation of the circle sector filter. Instead of letting the filter rotate around the image and read from different angles. All images were preprocessed: the images were rotated with a bicubic resampling filter. At every rotation step, a fixed circle sector filter stacked the input of the current pixel into a vector. The collection of vectors was then fed as input to a normal convolutional layer. The results presented below are not directly comparable with the other results in the report since the networks did not have the final design and were trained on a different dataset. This section in the appendix is included to strengthen the argument that the intuition of the specially designed R-CNN is a viable method but that the implementation could be reworked to enhance the performance.

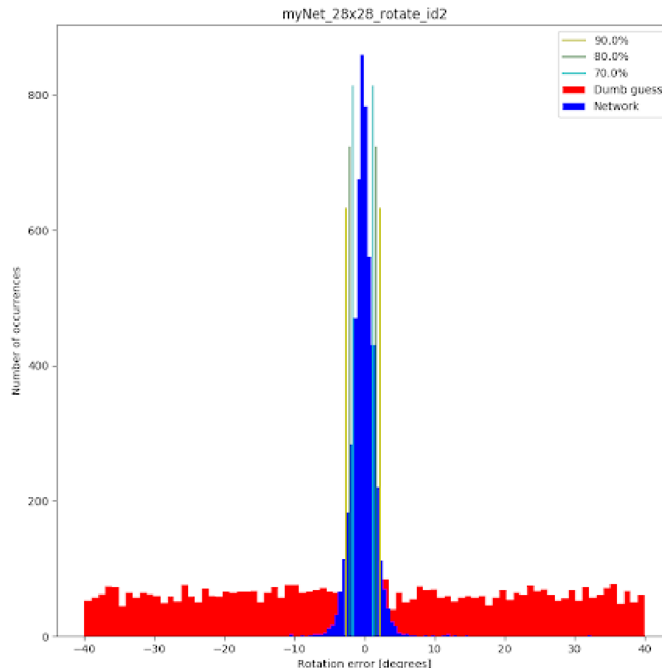

Figure A.4: Results on the rotation test set with one of the first models of the R-CNN using a preprocessing implementation for the specially designed rotational layer.

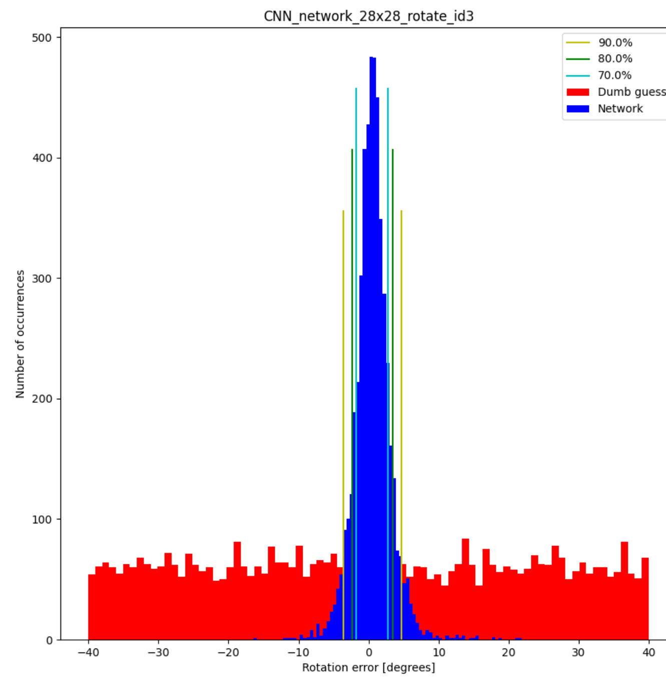

Figure A.5: A first baseline CNN performance on the rotational test set (constructed from the MNIST dataset).

A.3 Translation along only Y-dimension

Below result from translation tests along only the Y-dimension. The images in the dataset were only translated along the Y-dimension and the networks where trained for estimation on only the Y-dimension. In all other ways, these tests were designed in the same way as the other translation tests see section 4.1. One can see that the difference in performance between the two networks is small.

	T-CNN	Baseline CNN
mean |error|	0.21	0.08
median |error|	0.17	0.06
max |error|	3.13	5.52
validation loss	0.00063	0.00013
95% bound	[-0.51, 0.53]	[-0.18, 0.27]
90% bound	[-0.4, 0.43]	[-0.13, 0.21]
80% bound	[-0.31, 0.33]	[-0.08, 0.16]

Table A.1: Results from the translation test (only along Y-dim).

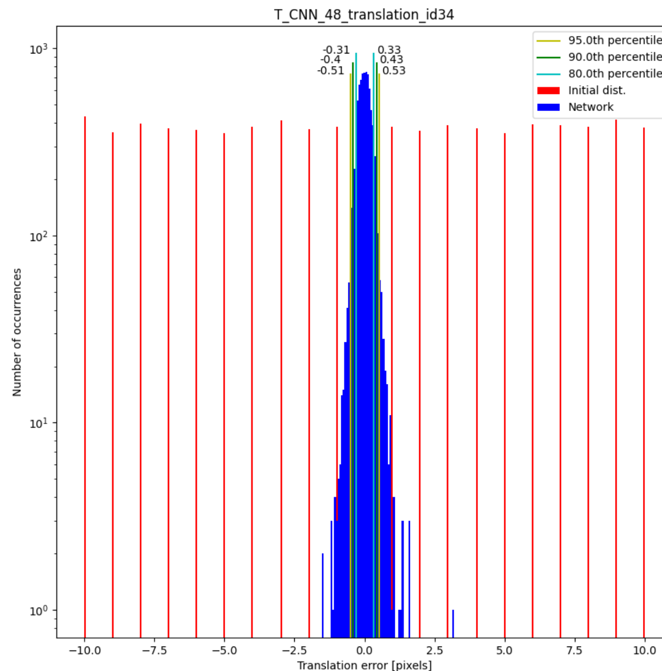

Figure A.6: Resulting histogram for the baseline CNN, translation along only Y-dim.

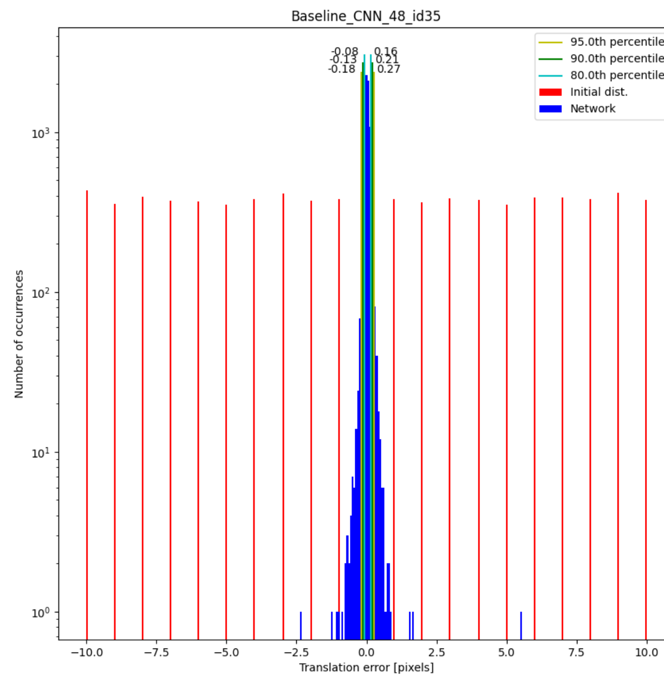

Figure A.7: Resulting histogram for the T-CNN, translation along only Y-dim.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY