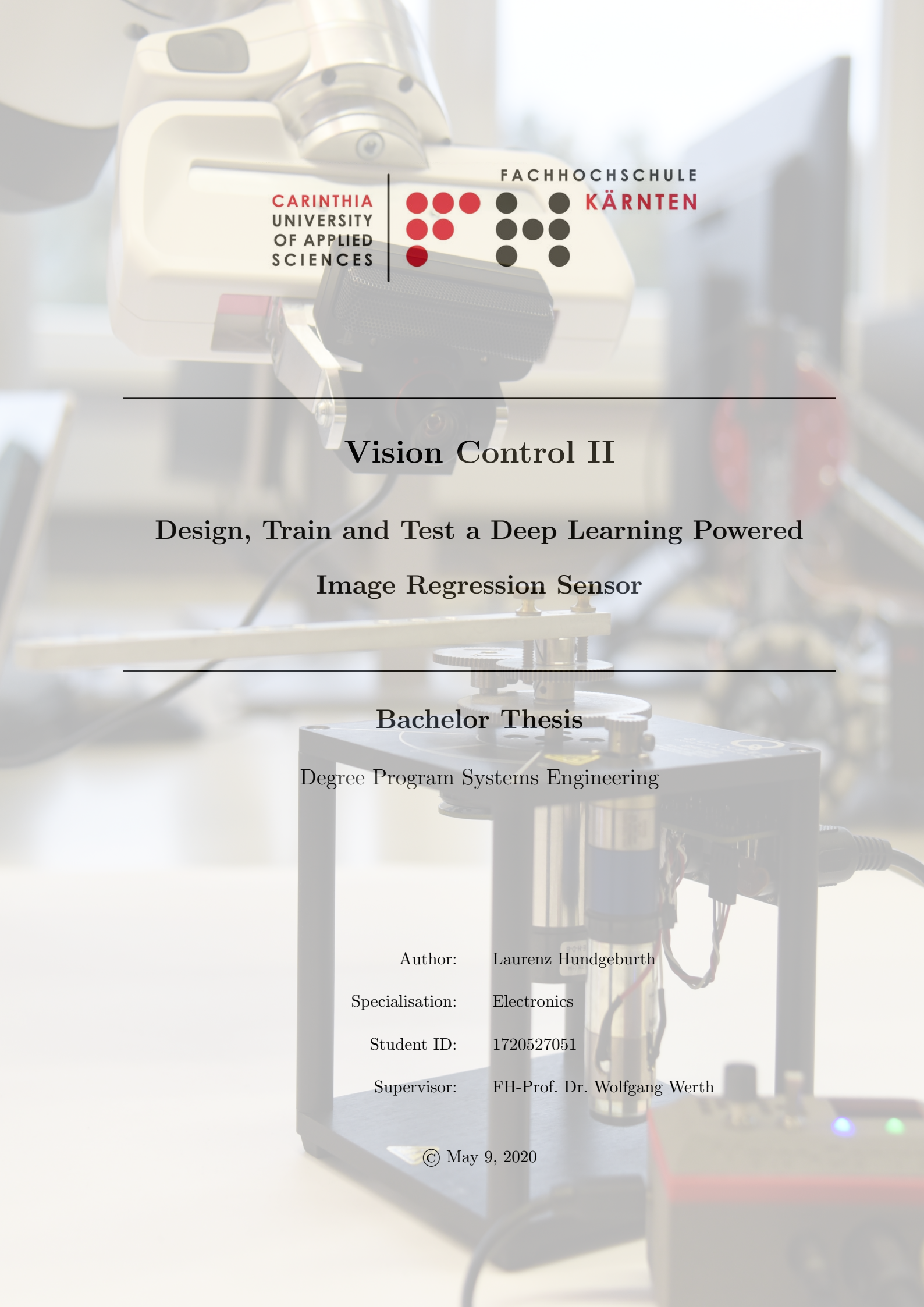


CARINTHIA UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE KÄRNTEN

Vision Control II

Design, Train and Test a Deep Learning Powered Image Regression Sensor

Bachelor Thesis

Degree Program Systems Engineering

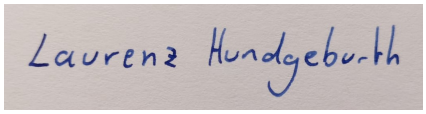
Author:	Laurenz Hundgeburth
Specialisation:	Electronics
Student ID:	1720527051
Supervisor:	FH-Prof. Dr. Wolfgang Werth

Statutory Declaration

I hereby declare that

- the Bachelor Thesis has been written by myself without any external unauthorized help and that it has not been submitted to any institution to achieve an academic grading.
- I have not used sources or means without citing them in the text; any thoughts from others or literal quotations are clearly marked.
- the enclosed Bachelor Thesis is the same version as the version evaluated by the supervisors.
- one copy of the Bachelor Thesis is deposited and made available in the CUAS library (Paragraph 8 Austrian Copyright Law [UrhG]).

I fully understand that I am responsible myself for the application for a patent, trademark or ornamental design and that I have to prosecute any resultant claims by myself.



Villach, May 9, 2020

Acknowledgment

First I want to thank my girlfriend Diana for it was she who always supported me in spite of having great difficulties understanding my enthusiasm for the subject.

I want to greatly thank both of my supervisors: professor Wolfgang Werth for showing great patience and trust in my capabilities by allowing me to pursue a topic close to my heart, and Heinz Peter Liechtenecker who proved to be an indispensable help and never ceased to amaze me with his programming and management skills.

It was professor Christoph Ungermanns who provided me with great advice yielding interesting results during the statistical analysis of my trained vision sensor.

I am very grateful for Andrew Ng's amazing online courses on Deep Learning [24] and Aurélien Géron's fantastic book "Hands-on Machine Learning" [10], both of which widened my horizon on the subject.

I also want to thank my friend and personal expert in all mechanical matters Nikolaus Gutenberger, who built us an enclosure for our motor control which surpassed all expectations.

Finally, I want to thank my project partner and close friend Gregor Fritz for always asking the right questions and for his willingness to either fail or succeed gloriously by my side.

Abstract

This thesis is based on a student project focusing on the study and application of Deep Learning powered Computer Vision. The ultimate goal was to design and train an image regression sensor able to predict the angular position of a servo motor shaft given only images as the sensors input. With the automated data acquisition of a large training data set being described in *"Vision Control I"* [9], this work describes the utilized Convolutional Neural Network architecture the vision sensor is based on, as well as the training settings and optimization algorithms used to train the sensor on the acquired data set. After the training and validation results have been presented with common Machine Learning methods including training curves and metrics, a more thorough statistical analysis of the trained sensor is conducted.

Keywords: Machine Learning, Deep Learning, Computer Vision, Image Regression, Convolutional Neural Networks, Optimization Algorithms, Statistical Analysis

Contents

List of Figures

List of Tables

1. Introduction

1.1. Overview

This thesis is based on a project which took place during the 4^{th} and 5^{th} semesters of the Systems Engineering Bachelor's degree programme at the Carinthia University of Applied Sciences (CUAS). The team consisted of my partner and colleague Gregor Fritz and myself, Laurenz Hundgeburth. We were kindly supervised by our professor Wolfgang Werth and our tutor Heinz-Peter Liechtenecker. During this project, we designed built and trained a Machine Learning (ML) vision sensor that can be used in our university's control laboratory.

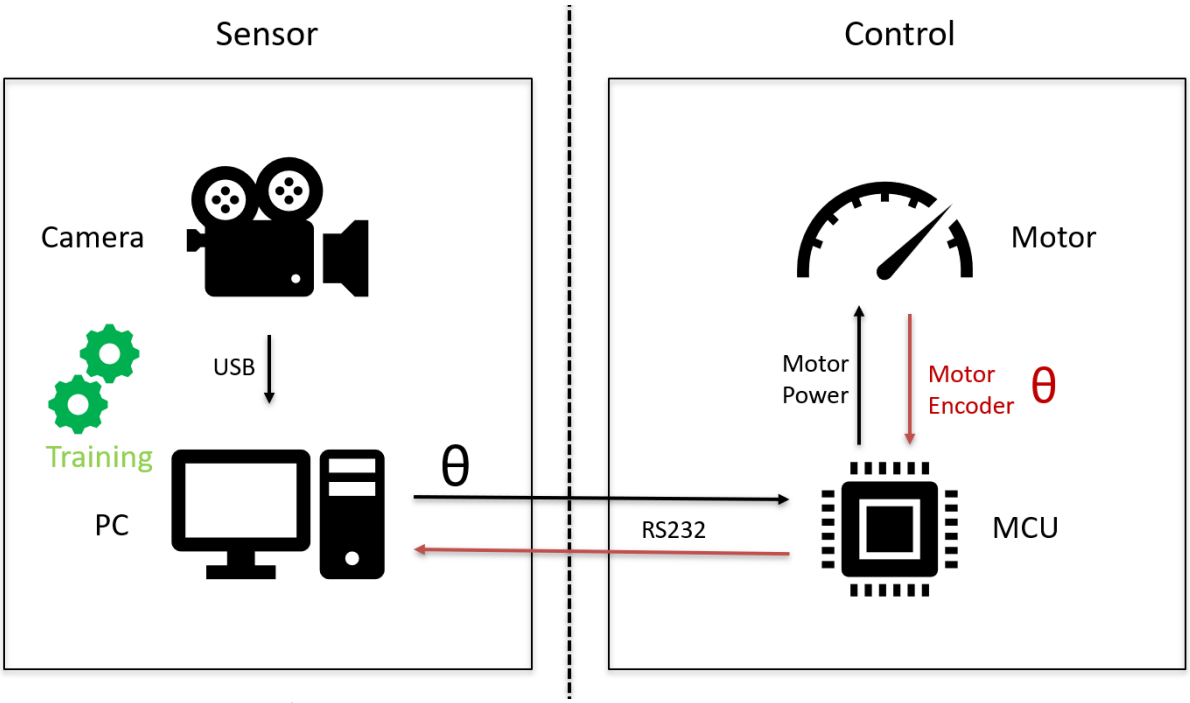


Figure 1.1.: Schematic of Experimentational Setup [9]

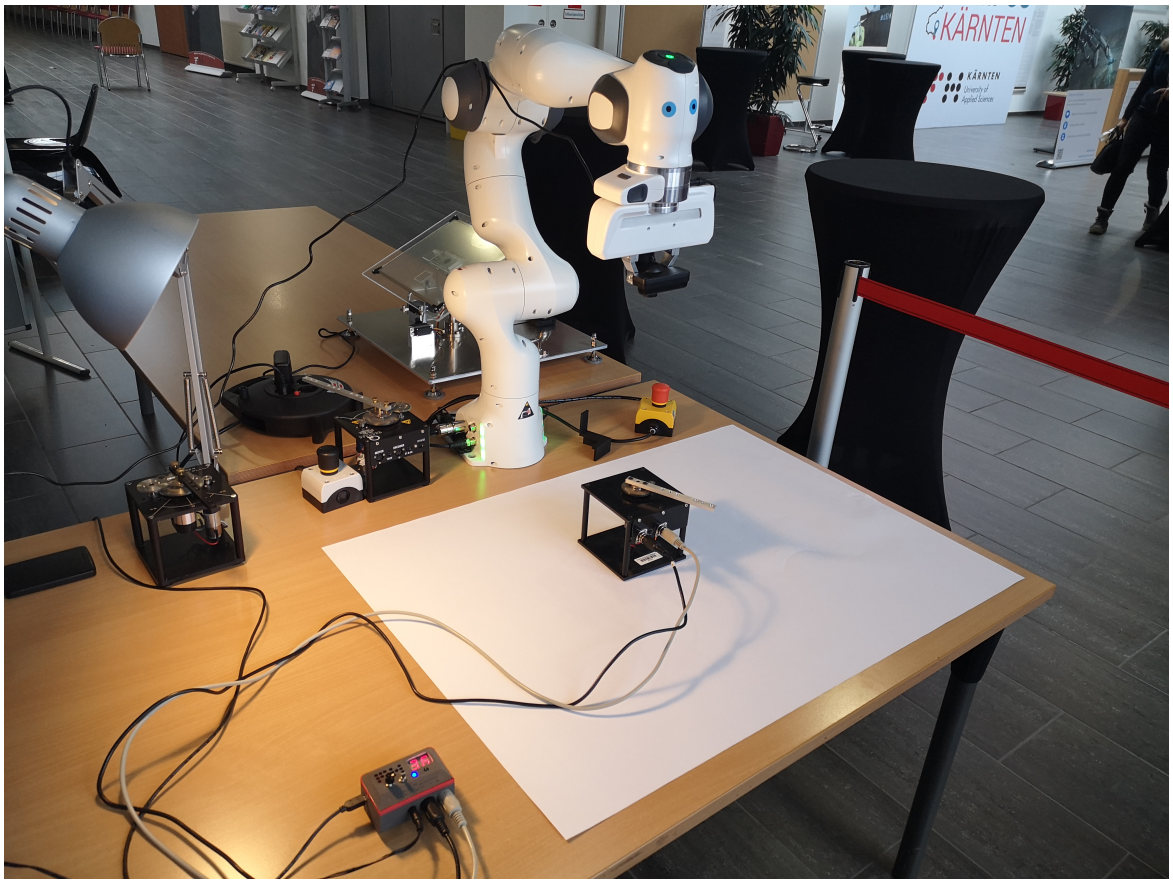


Figure 1.2.: Experimentational Setup [9]

The goal of this project is to predict the position of the motor shaft with a machine learning model.

A servo motor block from Quanser [27], which has a long shaft attached to its axis closely resembling an analogue clock or a speedometer, was chosen as the main actor. This servo motor has an optical 12-bit quadrature encoder enabling rotation measurements with a step size of 0.088°. This motor is controlled by a small custom Micro Controller Unit (MCU) which provides an interface from a desktop PC to the motor hardware. The last part of the experimental setup is a small PSEye Camera [26] mounted on a Franka Emika robot[8]. This camera, which is connected to the central PC via USB, takes images with a resolution of 480x640 pixels from the motor with a constant frame rate of 60fps.

The idea behind the setup is the automation of labelled data acquisition. Having an interface to both the camera and the motor allows the PC to save pictures with their current position values, as well as determining the next set point of the motor. With all parts communicating with each other no human intervention is required during acquisition. The robotic arm is constantly adjusting the position of the camera to get some variation in the camera's perspective into the data set.

The scope of this project can be roughly split into 6 parts:

1. Building and programming the control unit of the motor.
2. Designing an automated data acquisition system.
3. Acquiring a large data set required for training.
4. Selecting a ML model suited for the given task.
5. Designing and conducting the training of the vision sensor.
6. Test the sensor and discuss the results.

More information about the experimental setup and automated data acquisition system can be obtained in [9].

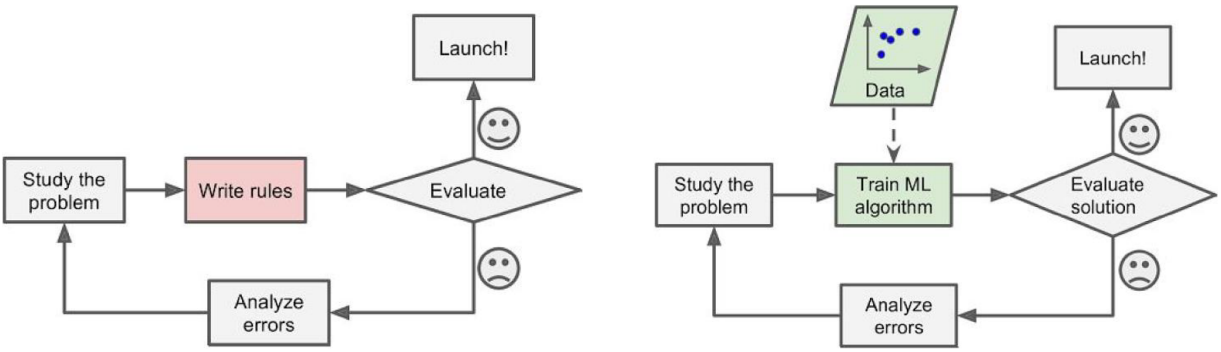
However, this work focuses on the training and evaluation of the vision sensor.

1.2. Motivation

One of an engineers most important skill is the ability to use mathematical models, which are able to describe and predict the behaviour of different phenomena. Because in order to control something, you first have to understand and describe it. When it comes to different engineering disciplines, there is a plethora of domain-specific knowledge and tools to tackle special problems. Often the origin of these tools and techniques precede the engineers using them by at least one

generation and are the product of thoughts and experiments. Due to their general nature, these simplified models tend to need a lot of adjustments and parameter tuning, to fit the needs of a specific use-case. In a typical engineering task, the actual model selection takes up significantly less time than obtaining and tuning all the different parameters needed for the engineer's solution to work as intended.

Having not to deal with this time-consuming part, an engineer would be able to invest considerably more time into the part of model selection and systems design. ML offers such an alternative approach as it gives 'machines' (computers programs) the ability to optimize themselves.



(a) Traditional Approach **(b)** Machine Learning Approach

Figure 1.3.: Iterative Problem Solving [10]

The motivation behind this work is to familiarize with the basics of Machine Learning and examine how some of its benefits can be leveraged in a control scenario by building a Convolutional Neural Network (CNN) based vision sensor with a very high dimensional input (images). Based on this work, the following research question will be answered in the discussion part in chapter 4:

- How can the labour-intensive work of building a ML powered Sensor be justified in different applications?
- What influence does the quality and amount of data has on the training results?
- How much background knowledge in Data Science Methods is needed for an engineer to be successful in ML projects?
- How suitable are ML based sensors for safety and/or performance-critical tasks?

1.3. Machine Learning - Overview/Idea

Ian Goodfellow framed the challenge to artificial intelligence incisively in his book 'Deep Learning': *"The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally — problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images."* [12]

Machine Learning offers a way to solve such problems. As with any mature study field, that developed over a long period into a variety of different sub-fields, there exists countless definitions, all trying to explain the same concept. To name just a few, here are three examples:

Machine Learning is the science (and art) of programming computers so they can learn from data [10].

[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed [Arthur Samuel, 1959].

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E [Tom Mitchell, 1997].

In Supervised Learning, a sub-field of Machine Learning this thesis concentrates on, the experience E comes from labelled training data consisting of a set of inputs and their corresponding outputs - the labels. The task T could be anything, but normally is either *Classification*, where we want to assign the input to a certain class (e.g. Spam or Ham Emails), or *Regression*, where we want to predict a continuous value from the given input (e.g. predict the value of a car given some information about it).

The idea behind Supervised Learning is to train a mathematical model on the given training data, so that it eventually learns the relationship (hypothesis) between input and output and is then able to use this "learned knowledge" to predict on new input data, the model has not seen during training [17].

2. Methods

2.1. Machine Learning - Introduction

2.1.1. Supervised Learning

After the careful high-level introduction to Machine Learning in section 1.3, there is a need to frame the problem mathematically. A lot of notation used throughout the thesis will be introduced here.

The goal of any Supervised Learning Project is to find a mathematical model describing the relationship between the chosen inputs and outputs.

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m \qquad [12] \tag{2.1}$$

n represents the number of dimensions of the input space, whereas m represents the number of dimensions of the output space. In the case of the project, this thesis is based on, n is the number of pixels times the number of colour channels from the input images $:= 480 * 640 * 1 = 307.200$ (1 channel since we use grayscale images, see 2.2.3), and $m = 1$, as only a single value - the angle of the motor shaft - is predicted.

$$h(X) = \hat{y} \qquad [24] \tag{2.2}$$

Equation (2.2) is a common way to frame our problem. X is a multidimesnional input - in our case an image. $h(X)$ is the hypothesis - a ML term used to describe the mathematical model making the predictions. In other ML books and papers, it is often substituted with the simpler and shorter $\hat{y}$.

$h(X)$ can have many forms, from simple linear functions to high order polynomials, or even deep Artificial Neural Network (ANN). Choosing a good model for a given task is one of the main tasks in ML projects and is subject to many research groups. The details of the model chosen for the problem of the thesis will be explained in section 2.2.4.

Regardless of the form of $h(X)$ it has a variety of model parameters (like the slope and the offset of a linear function). These parameters describe the behaviour of the model. "Finding" the right parameters which lead to good representations is the goal of any ML training process. There exists a variety of different parameter initialisation strategies, but the details are not important to understand the concept. With a model having its parameters initialised "randomly" it will probably perform very poorly on a chosen task. The "finding" of the right parameters then basically consists of many training steps, in which the parameters get slightly adjusted to improve the model's accuracy.

For ML models to know how to adjust their parameters, they need labelled training data. Generally speaking, the bigger the model and the more complex the task, the higher the amount of available training data the model needs to converge to an acceptable solution [14]. We will discuss the amount of data needed for this project in section 3.3.

2.1.2. Loss Optimization

The optimization of the model's loss $J(X)$ is one of the most important aspects of ML. It is the one part responsible for the improvement of the model's accuracy during training. But we need to define the term "loss" first. The loss function is simply the measure of the error of the model's predictions over some input [21]. It measures how poorly a model performs on a given task. There

exist countless loss functions which specialise on different problems, but a loss can be as simple as the Mean Absolute Error (MAE) (see 2.2.5.)

$$\underset{W}{\text{minimize}} \quad J(W, X) \quad , where \quad W...ModelParameters \tag{2.3}$$

Minimizing this loss over some training data would *maximize* the model's accuracy. Some functions define how well a model performs. Such functions are called objective functions and should be maximized instead of minimized, but they are rarely used in Supervised Learning [21].

Finding the global minimum of a loss function, however, is not trivial, as they depend on every parameter of the model. One solution is to use numeric optimization strategies and approach the global minimum iteratively. The iterative nature of these optimization processes is one of the reasons why training a ML model takes considerably longer than comparable modelling techniques.

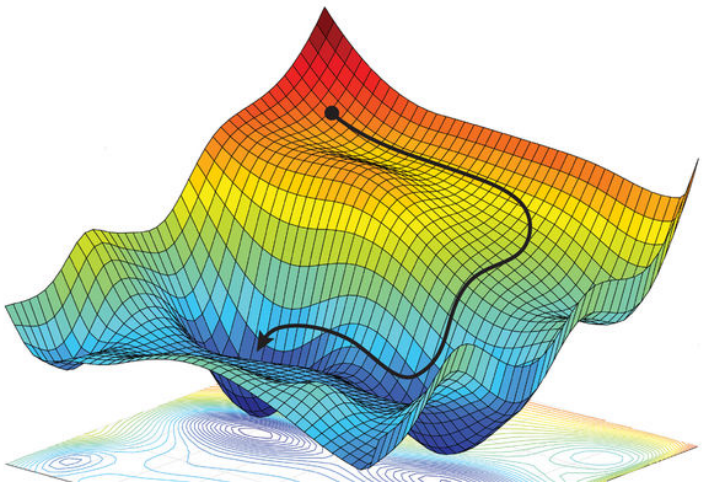


Figure 2.1.: Loss Minimization Conceptually [15]

Most of these numeric optimization techniques are gradient-based, were for each training step, we need to know the negative gradient of the loss function with respect to every parameter: $-\frac{\partial J(W,X)}{\partial w_i}$. Having this information, we know in which direction and how strong we need to adjust w_i for this training step to have a maximal impact on the loss minimization.

These gradients can then be used in optimization algorithms like Stochastic Gradient Descent (SGD) or Adaptive Momentum (ADAM), the latter of which will be explained in section 2.2.6.

2.1.3. Generalization

Simply minimizing the training loss does not necessarily yield a good representation, as we are not only interested in the performance on the training data, but more so in the model's capability to apply its learned knowledge to new examples [12]. Speaking in ML terms, we are interested in the model's ability to generalize well. Depending on the model's architecture and complexity, bad generalization can be caused by under-fitting (high bias) or over-fitting (high variance).

Under-fitting occurs if the model complexity is too low for the given task. A linear model, for example, does often not suffice in representing high order polynomials or nonlinear functions well. On the other hand, if the training set includes a considerable amount of noise or has too few examples, a complex model may learn patterns that do not represent the origin of the data.

To monitor generalization during training it is very helpful to have a second data set the model is not trained on. This validation set is only used to compute the loss and other metrics over novel examples. Plotting the loss of both the training and the validation set over the training steps (learning curves) gives a very good overview of both, the model's performance and its generalization capabilities.

In practice models tend to become more and more complex, making over-fitting a more common problem to take care of than under-fitting. Measures against over-fitting will be discussed in section 2.2.

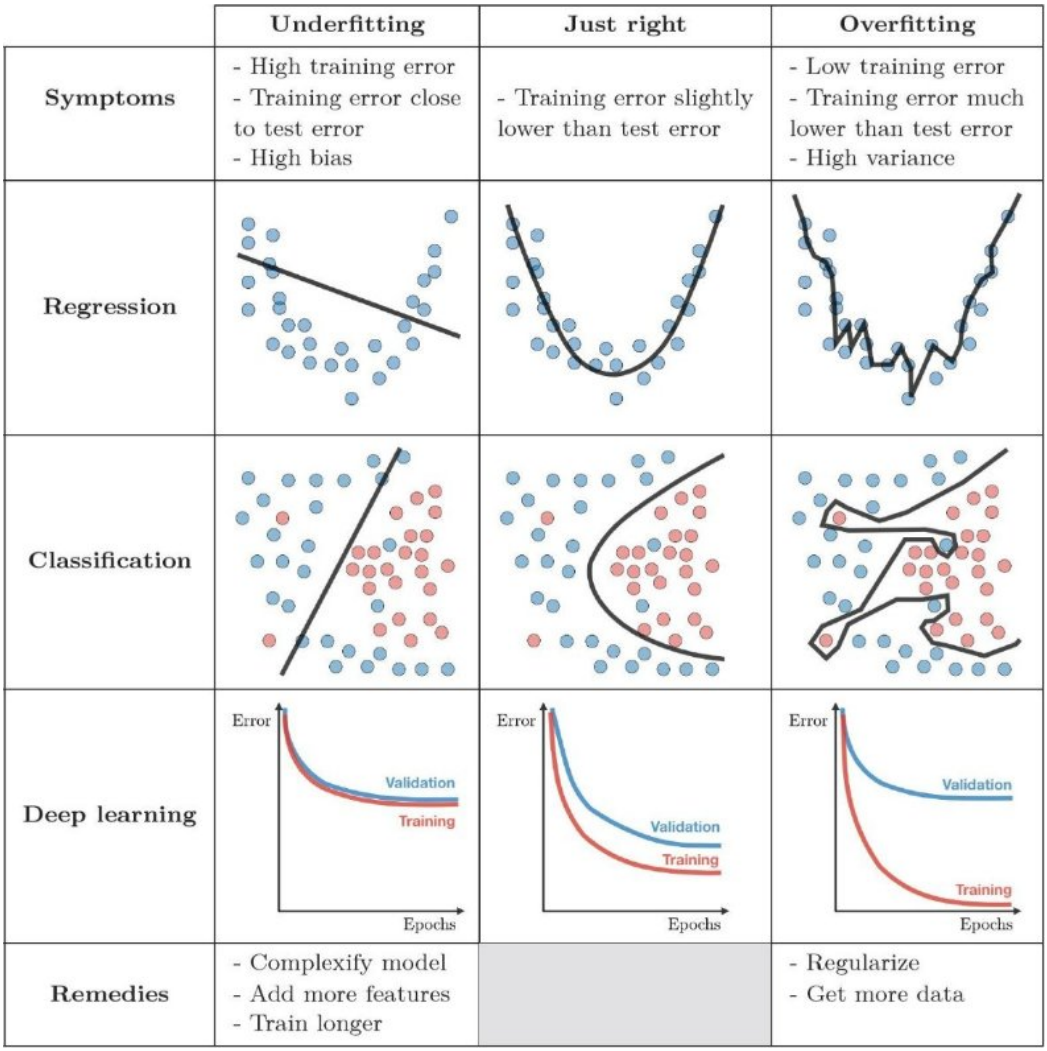


Figure 2.2.: Generalization Overview [3]

2.2. Training of the proposed CNN

2.2.1. Hardware and Software

Modern tools like deep learning frameworks and libraries greatly reduce the complexity of ML projects, as they not only have many easy to use implementations of state-of-the-art algorithms, but they often take care of the complete training process and optimization of the chosen model. It is due to their ease of use and performance, that using such frameworks has become the standard in AI research.

One of such tools is Google's deep learning framework TensorFlow [22]. In TensorFlow models are described using computational graphs which allows TensorFlow to implement the Reverse-Mode Automatic Differentiation algorithm. The implications of this are that there is no need to calculate the gradients of the loss function by hand, which can get very time consuming and complex. Another big advantage when using TensorFlow is that in case of an unexpected training outcome, errors and bugs in the training algorithm are not to be expected which greatly speeds up development and research.

However, leveraging TensorFlows advantages requires the translation of a mathematical model into a computational graph understandable by TensorFlow. To further decrease the burden of entry and empower as much ML progress and research as possible, TensorFlow provides a high-Level Application Programming Interface (API) abstracting away the low-level implementation details. Using such a high-level API requires very few functions and API calls to execute complete training processes. Furthermore, it offers easy to use building blocks for designing ML models which then internally get transformed into computational graphs. The high-level API used in this project is called Keras [2], which only recently became TensorFlows official high-level API.

The standard choice of a programming Language in most ML projects is Python [29]. It is a beautiful, high-level interpreted scripting language that focuses on readability of code and simplicity. This beauty and simplicity come at a performance cost which can be circumvented by using pre-compiled, highly optimized libraries (like TensorFlow) for the performance-critical tasks like optimization workloads.

When working in ML projects, it is very important to not neglect the hardware requirements of Deep Learning training routines. While on common hardware simple models can be trained on small data sets in mere seconds to minutes, it took 4 days to train googles newest Natural Language Processing model BERT on 16 Tensor Processing Pods having a total processing power of approximately 1.600 PetaFLOPS [5].

In this project, GPU hardware acceleration was used and training our model on an NVidia RTX 2080 Ti [39] (3.600.000 training steps, 80.000 480*640 pixel images, CNN model with approximately 220.000 parameters) took about 2.5 days.

2.2.2. Data Exploration

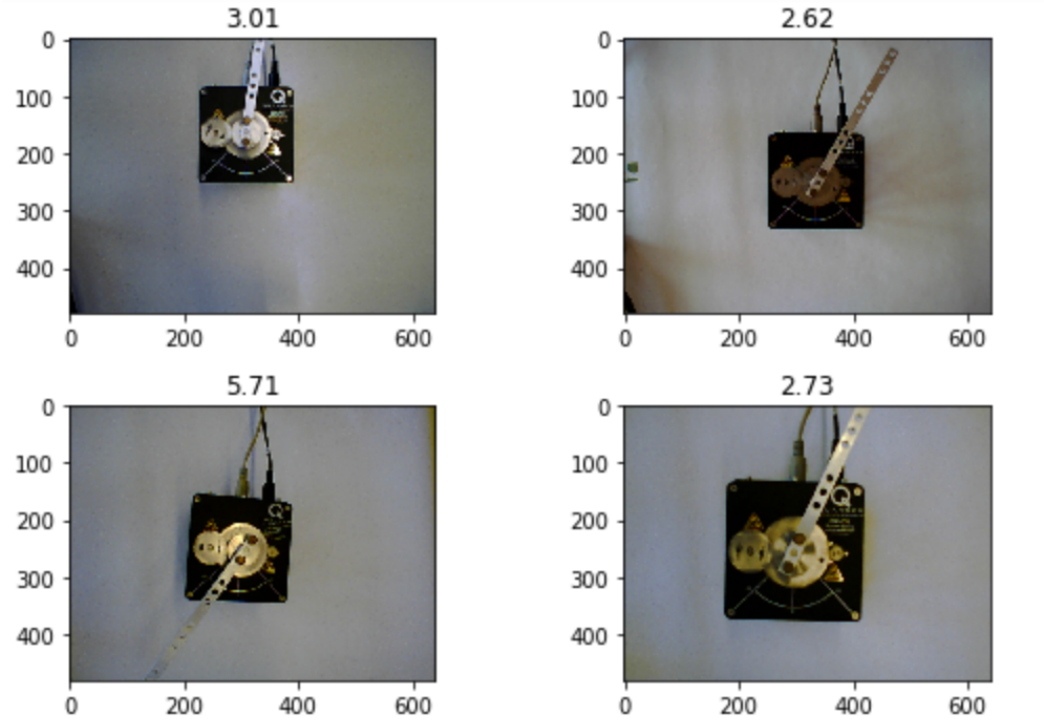


Figure 2.3.: Training Set Image Samples

During the Bachelor Project, a training set consisting of 80.000 labelled images and two test sets consisting of 10.000 labelled images each were acquired. A detailed description of the acquisition method and many thoughts and considerations regarding the experimental set-up can be found in [9].

All three data sets come in the form of a directory containing all the images and a Comma-Seperated Values (CSV) File with two columns. One column contains the filenames of all the images inside the corresponding directory, and the other column contains the corresponding labels.

This data set format is called a DataFrame [7]. Such a data frame can be easily loaded into TensorFlows' high-level API Keras. A common alternative would be to store the whole dataset inside one big multi-dimensional matrix which gets loaded into the PC's Random Access Memory (RAM) during training, provided that the used PC has enough RAM. The training set size alone amounts to 28,3 GB this approach was not applicable. The downside of the DataFrame approach is that for each training step the training instance has to be loaded from disc, which is slower than accessing it from RAM.

2.2.3. Data Augmentation and Preprocessing

Loading each training image from the computers hard drive does have the advantage of enabling a separate data augmentation step without having to touch the original data set. Data Augmentations include any steps, that augment or change the original data, without changing the semantic/contextual meaning of the data. These augmentations have a very positive impact on both the trained model's accuracy and its generalisation capabilities, as they effectively increase the amount of training data - one of the most critical aspects of ML projects [23]. Preprocessing steps are used to reduce the dimensionality of the inputs, thus requiring smaller models to learn the input-output relationship. Image Augmentations and preprocessing steps used in this project include:

- **Color Transformation from RGB to Grayscale** - reduces the amount of inputs by a factor of 1/3 (921.600 $\rightarrow$ 307.200). This is only applicable if the desired input-output connection does not depend on color information.
- **Re-scaling by a Factor of 1/255** - as each pixel value ranges from 0 to 255 (8 bit) dividing the input images by 255 scales every input pixel to a range of 0 - 1. This is common ML practice as it brings the input values nearer to the activation functions non-linearities, thus helping to improve accuracy and convergence time [19].
- **Width/Height Shifting by up to 25 Pixel** - simulates camera movement in the x-y plane

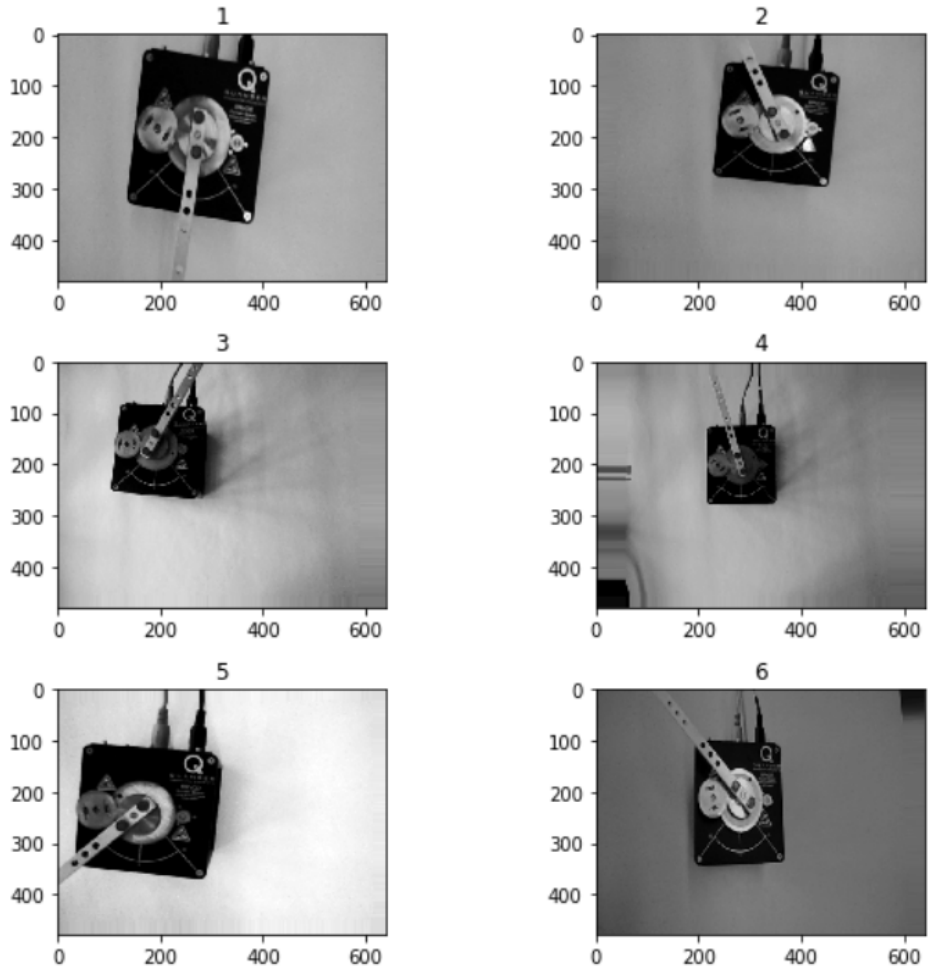


Figure 2.4.: Image Data Augmentation Samples

- **Zooming by up to 25%** - simulates camera movement along z-axis
- **Rotation by up to 5°** - simulates camera rotation
- **Brightness Shifting between 70% and 130%** - increases the models performance under different lighting conditions

Data Augmentation strategies and their parameters should be carefully selected as they can easily distort the image in such a way, that its semantic meaning gets lost. Shifting the width of the images from Figure 2.3 too much to the left or right, for example, would results in images without the centre of the motor which is crucial in measuring the angle of the motor shaft. Rotating the

images by more than 45° would lead to a loss of the assumption, that the reference point of the angular measurements is always facing the bottom corner of the image.

2.2.4. Model Selection

Before the model architecture chosen for this project will be explained in section 2.2.4.7 - **Model Architecture**, its basic building blocks will be explained in more detail to have a better understanding of the final mathematical model.

2.2.4.1. Densely Connected Neural Networks

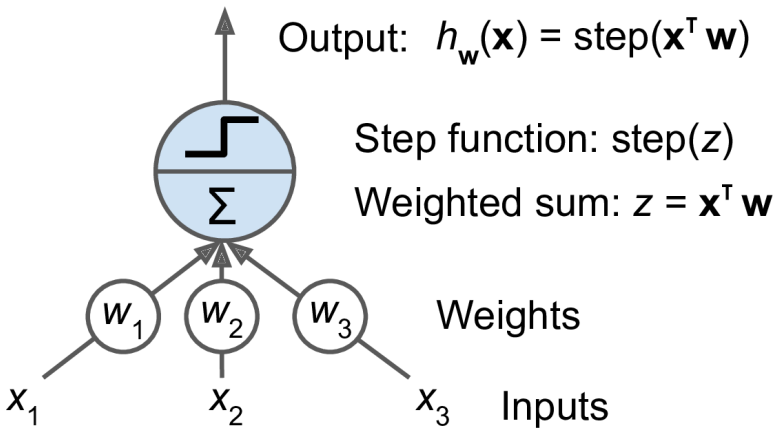
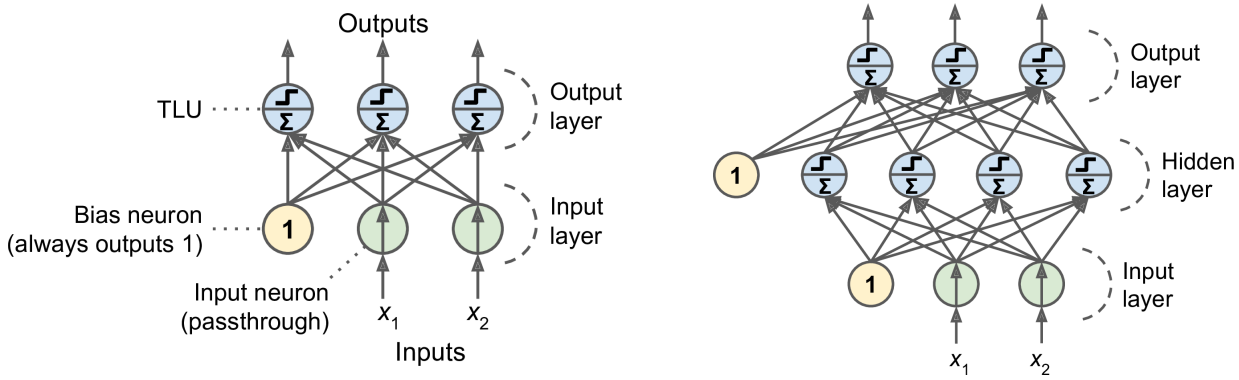


Figure 2.5.: Rosenblatt Perceptron 1957 [10]

The Perceptron invented by Frank Rosenblatt in 1957 is one of many attempts in mathematically modelling the behaviour of a biological neuron. [28] Compared to other models, the perceptron is especially simple and computationally cheap which is one reason why it was a very popular ML research subject. The Perceptron consists of a step function of the sum of weighted inputs, which resembles the ”firing” of a biological neuron when its activation potential surpasses a certain threshold. The weights represent how strong the inputs contribute to the activation of the corresponding neuron.

Composing layers of Perceptrons with every perceptron connected to every input creates *fully connected* or *dense* layers. Because of their linear operations, perceptrons were unable to learn



(a) Perceptron with two input neurons, one bias neuron, and three output neurons [10]

(b) Architecture of a Multilayer Perceptron [10]

Figure 2.6.: Perceptron Networks [10]

non-linear problems like the XOR operation, which is one of the reasons perceptrons and neural networks lost some popularity until in 1986, David Rumelhart, Geoffrey Hinton, and Ronald Williams proposed a way to train multilayer networks using the backpropagation algorithm. [31]

Adding Bias inputs and more capable activation functions (see section 2.2.4.2 - **Activation Functions**) neural networks can learn very complex functions. [12] This is why they are widely used in many different ML projects. Their only problem is their high number of model parameters when there are a lot of inputs and/or the model has several hidden layers.

We will overcome this issue by first reducing the dimensionality of our input images through the means of convolutional layers and pooling layers both of which are described in sections 2.2.4.3 - **Convolutional Layers** and 2.2.4.4 - **Pooling Layers**.

2.2.4.2. Activation Functions

Activation functions can be found in the simplest ML project. Broadly speaking, they mainly have three purposes:

- Bringing the output of a model in the desired shape. A linear function as a models output can be used in regression problems, whereas in probability estimation, we want an output

S/N	Function	Computation Equation
1	Sigmoid	$f(x) = \left(\frac{1}{(1+exp^{-x})} \right)$
2	HardSigmoid	$f(x) = max\left(0, min\left(1, \frac{(x+1)}{2}\right)\right)$
3	SiLU	$a_k(s) = z_k \alpha(z_k)$
4	dSiLU	$a_k(s) = \alpha(z_k)(1 + z_k(1 - \alpha(z_k)))$
5	Tanh	$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right)$
6	Hardtanh	$f(x) = \begin{pmatrix} -1 & if\ x < -1 \\ x\ if & -1 = x \leqslant 1 \\ 1 & if\ x > 1 \end{pmatrix}$
7	Softmax	$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$
8	Softplus	$f(x) = \log(1 + exp^x)$
9	Softsign	$f(x) = \left(\frac{x}{\|x\|+1} \right)$
10	ReLU	$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$
11	LReLU	$f(x) = \alpha x + x = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases}$
12	PReLU	$f(x_i) = \begin{pmatrix} x_i\ , & if\ x_i > 0 \\ a_i x_i\ , & if\ x_i \leq 0 \end{pmatrix}$
13	RReLU	$f(x_i) = \begin{pmatrix} x_{ji}\ , & if\ x_{ji} \geq 0 \\ a_{ji} x_{ji}\ , & if\ x_{ji} < 0 \end{pmatrix}$
14	SReLU	$f(x) = \begin{pmatrix} t_i^r + a^r{}_i (x_i - t^r{}_i), & x_i \geq t^r{}_i \\ x_i\ , & t_i^r > x_i > t_i^l \\ t^l{}_i + a^l{}_i \left(x_i - t^l{}_i\right), & x_i \leq t^l{}_i \end{pmatrix}$
15	ELU	$f(x) = \begin{pmatrix} x, & if\ x > 0 \\ \alpha\ exp(x) - 1, & if\ x \leq 0 \end{pmatrix}$
16	PELU	$f(x) = \begin{pmatrix} cx, & if\ x > 0 \\ \alpha\ exp^{\frac{x}{b}} - 1, & if\ x \leq 0 \end{pmatrix}$

Figure 2.7.: List of Common Activation Functions [25]

between 0 and 1 for which the logistic function or sigmoid function is commonly used. There are also activation functions to deal with multi-class classification problems like the SoftMax function. [25]

- Using non-linear activation functions enables the model to learn complex, non-linear relationships which even in deep neural networks would be impossible if only linear activations are used. [25]

- They help control the numerical properties of the optimization process. If chosen uncarefully, they can introduce problems like vanishing gradients, where the backpropagated error

gradients get smaller and smaller which slows down convergence and has a negative impact on accuracy. [4].

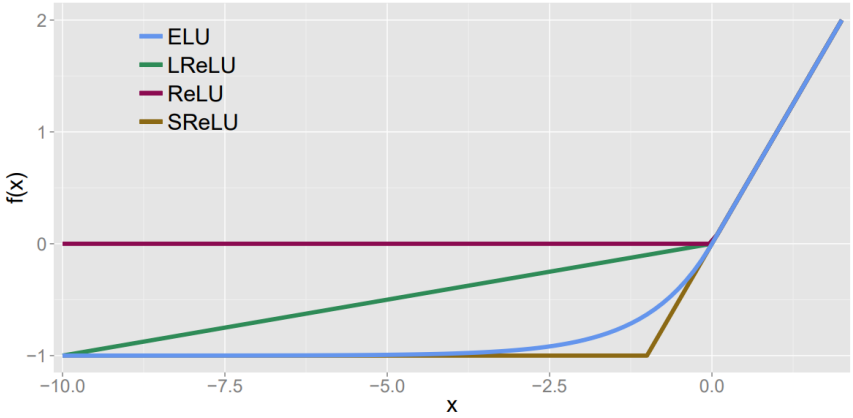


Figure 2.8.: Linear Unit Activation Functions [4]

As the output for the given regression problem, a ReLU function ($f(x) = max(0, x)$) has been chosen for the model to have a linear output, while at the same time guarantying positive predictions.

For the hidden layers of the the ELU function (equation 2.4) with an α value of 1 has been chosen for its excellent numerical properties. It brings the mean activations closer to 0, while still avoiding vanishing gradient problems with its linear unit [4].

$$f(x) = \begin{cases} x, & if \;\; x_i > 0 \\ \alpha \exp(x) - 1, & if \;\; x_i \leq 0 \end{cases} \tag{2.4}$$

2.2.4.3. Convolutional Layers

The biggest reason why densely connected Neural Networks perform poorly on big images is the exploding numbers of model parameters when using deep models.

One of the biggest breakthroughs in Computer Vision was when in 1999 Yann LeCun proposed to use Convolutional layers in image recognition tasks in his paper about the LeNet architecture [20].

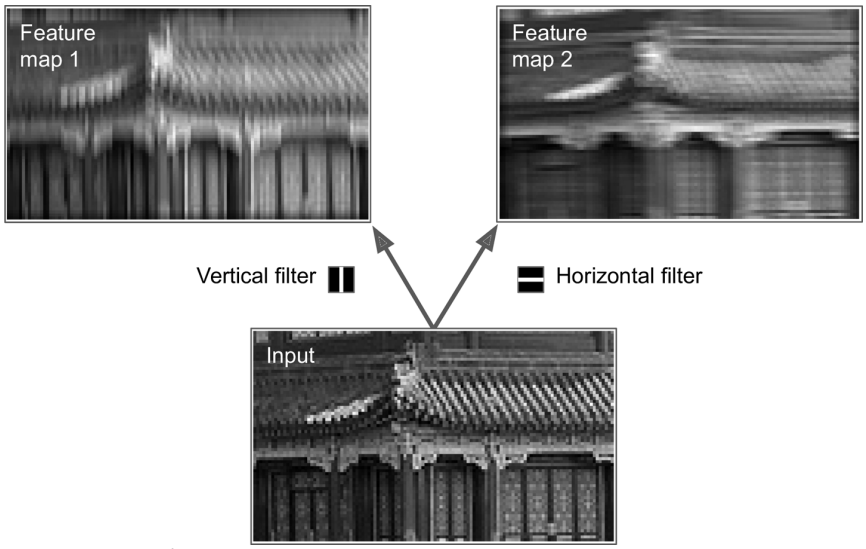


Figure 2.9.: Applying two different Filters [10]

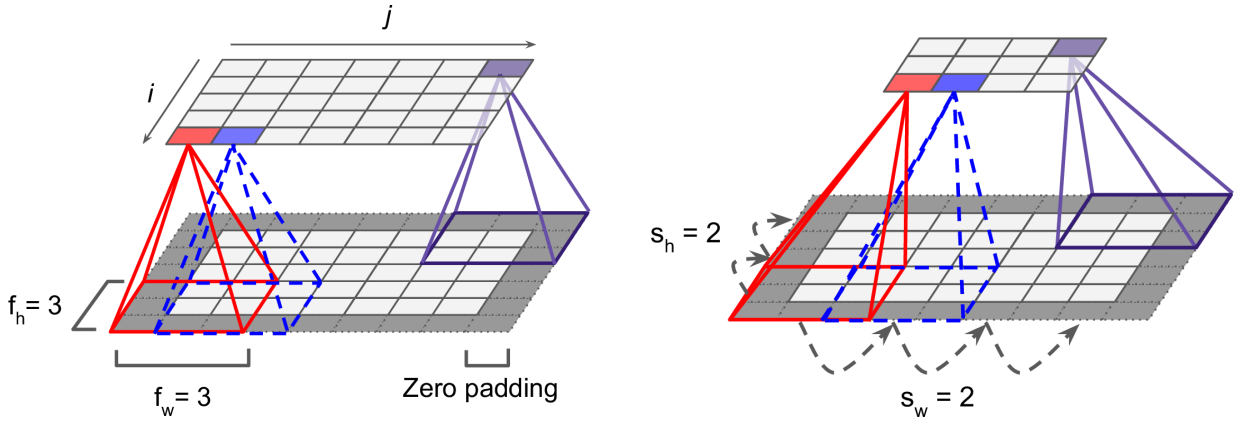
The idea is to restrict the receptive field of a neuron to a small field containing only a handful of pixels (often 3x3 or 5x5). By stacking multiple layers of those "restricted" neurons, low-level features can be extracted in the first layers, which then get assembled into higher-level features in the following layers [10].

By reusing the weights of such a small "restricted" neuron in all neurons of the same layer, a huge amount of parameters can be saved, allowing to have very deep networks with very few parameters compared to densely connected Neural Networks.

Mathematically this behaviour can be modelled with the convolution operation. Though technically most ML project use cross-correlation due to its simpler implementation, referring to this operation as "convolution" has prevailed. the difference lies only in the flipping of the used kernels in the convolutions [12].

Because convolutions have been used on images in tradition image processing before, some terminology has been adopted into CNN descriptions:

- The weights of a neuron restricted to a receptive field inside a convolutional layer are referred to as **Filters** or **Kernels**. Just like any other parameters they are determined during training and can be visualised with small images.



(a) Keep dimensions using Zero Padding and a stride of 1

(b) Reducing dimensionality using a stride of 2

Figure 2.10.: The Effect of different stride sizes [10]

- The output images of convolutional layers are referred to as **Feature Maps**

It is common practice to use multiple filters inside one convolutional layer to obtain a stack of feature maps with each feature map belonging to a certain filter. Using RBG images like in Figure 2.9 can be interpreted as starting with three feature maps. However, when chaining multiple convolutions, the input feature maps are more commonly referred to as image channels.

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad with \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \qquad [10] \qquad (2.5)$$

In Equation 2.5:

- $z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer (layer l).

- s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field, and f_n is the number of feature maps in the previous layer (layer $l\breve{}1$).

- $x_{i,j,k}$ is the output of the neuron located in layer $l\breve{}1$, row i, column j, feature map k.

- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k.

- $w_{u,v,k,k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k.

[10]

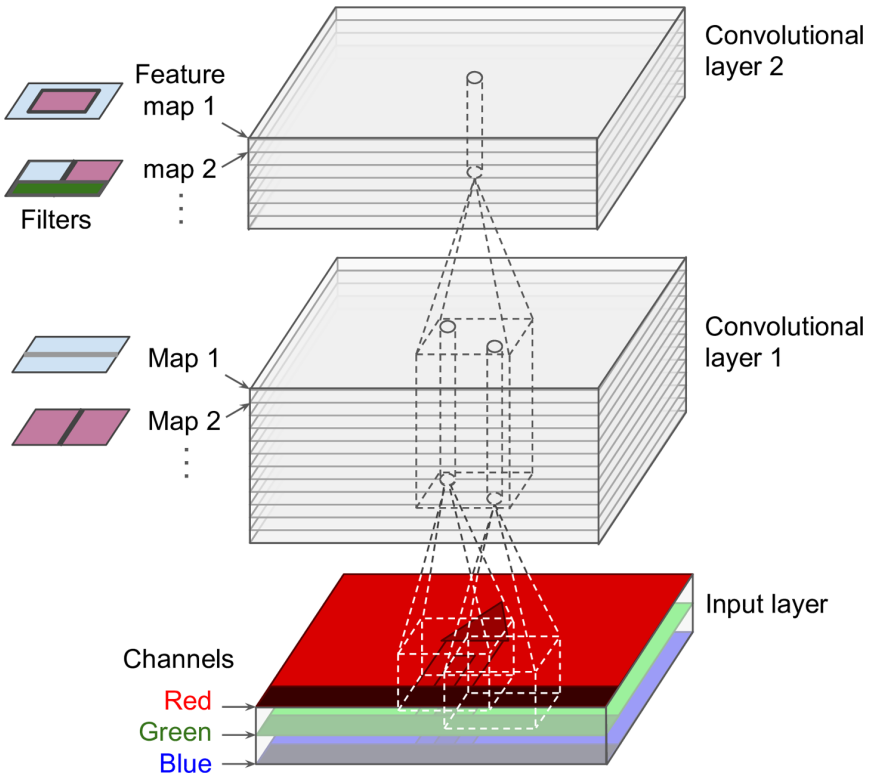


Figure 2.11.: Convolutional Layers with three Color Channels [10]

The larger the kernel size and the larger the stride size, the smaller the feature map will become. In deep CNN this can reduce the size of the feature maps dramatically. Often it is desired to have more control over the size of these feature maps with means like pooling layers described in section 2.2.4.4. Therefore many ML projects use Zero-Padding, where the image or feature map gets padded with zeros, artificially increasing its size so that the reduction of size through the followed convolution resizes the image or feature map back to its original size.

2.2.4.4. Pooling Layers

When dealing with high-resolution images, it is important to keep the inherent redundancy of information in mind. Often large image sections belong to the same group or object and do not contribute new or important information. Nevertheless, they do increase the total amount of input values and model parameters, thus requiring more calculations. It is therefore only natural to use downsampling methods like pooling layers to increase computational performance, while at the same time increasing shift-invariance. [34] Other sources also proclaim the high impact of pooling layers on the model's generalisation capabilities, as fewer model parameters leave the model less room to over-fit the training data. [33]

The most common downsampling methods in CNN are Max Pooling layers and Average Pooling layers. In both layers, a window or kernel with a certain size slides over every channel independently with a certain stride and outputs a single value per stride, which either is the biggest value (Max Pooling) or the average value (Average Pooling) of the kernel. [10]

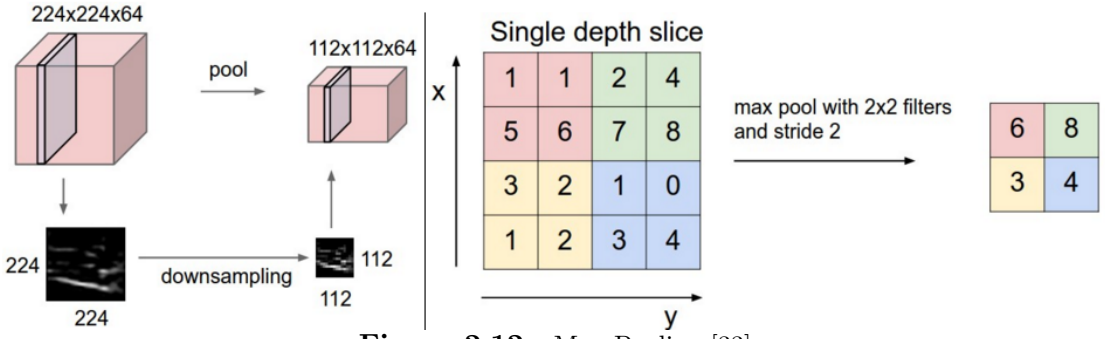
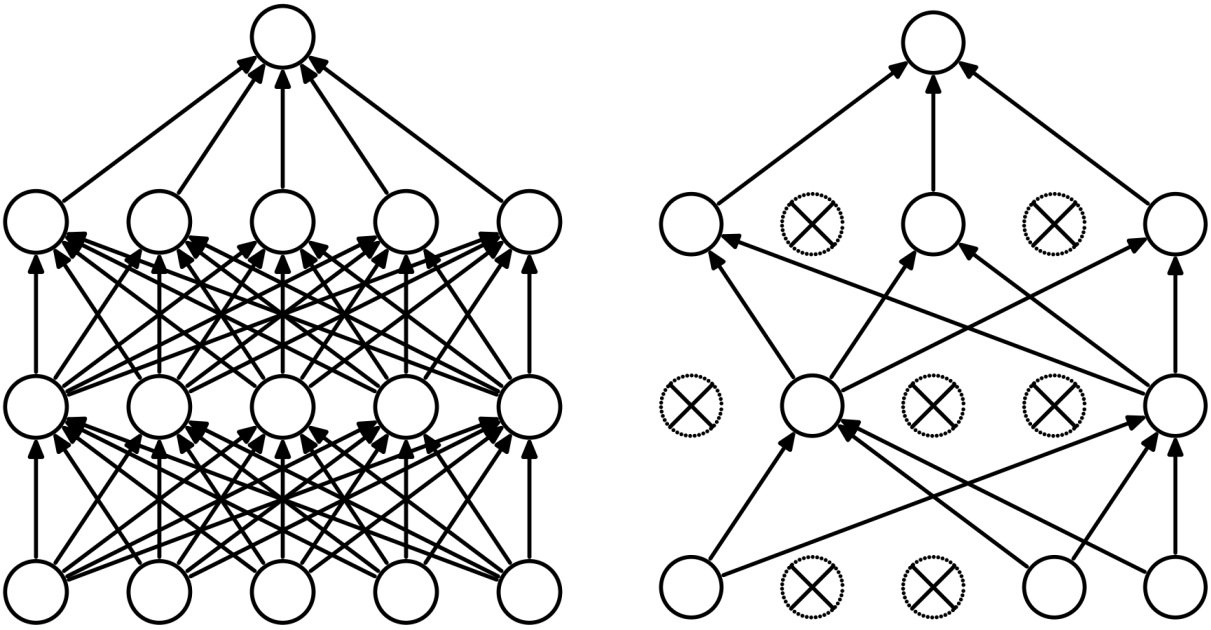


Figure 2.12.: Max Pooling [33]

Another common form of pooling found only in the last spacial layers of a CNN is the Global Average Pooling layer, which is an Average Pooling layer with a window size equal to the total channel size. This layer has the very useful effect of enabling CNN to accept different sizes of input images, as the convolution and pooling layers before the Global Average Pooling layer are not bound to a certain input size and the size critical dense layers after the Global Average Pooling layer receive a constant size output to perform their calculations on [10].

In this project, both Global Average Pooling and Average Pooling with a kernel size of 2*2 and a stride of 2 are used. Max Pooling has also been tested but performs slightly worse than Average Pooling for our problem.

2.2.4.5. Dropout Layers



(a) Neural Network **(b)** Applying Dropout

Figure 2.13.: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [35]

.

As discussed in section 2.1.3 over-fitting can become a problem when training deep neural networks due to their large size and many parameters. Another common problem is that without any regularization they tend to focus on a few numbers of specific features which are specific to the training set they have been trained on. Using generalisation techniques, they are forced to look at a wider range of features to make their predictions [10].

One very popular regularization technique is Dropout. During training, a random, predefined percentage of neurons and their connections get dropped resulting in a thinner version of the

original network. This forces the network to look for different features when the more obvious features get dropped. It is important to scale the activations during test time when dropout is no longer used because of the increased amount of neurons [35].

2.2.4.6. Batch Normalization Layers

Before talking about Batch Normalization, we need to clarify the term Batch. When training ML models, there are three approaches to obtain the error gradients for each training step:

- In **stochastic** training, the error gradients are computed for each input individually.
- In **batch** training, the error gradients are computed for every training input and the mean of all those gradients is then used to optimize the model. These averaged gradients generally point more to a global optimum suitable for all inputs, than the stochastic version, ultimately leading to a better solution. However, this comes at the disadvantage of having to compute the gradients for each training input in every training step, which can significantly slow down training when using large data sets.
- **Mini batch** training compromises on this idea by using batches with sizes smaller than the training set size (often between 8 and 32).

During the training deep Neural Networks, the changing input distribution of each layer can vary considerably requiring the use of smaller learning rates and careful parameter initialization strategies to counteract this fact, which will slow down the training. In their paper from 2015, Sergey Ioffe and Christian Szegedy refer to this problem as the *internal covariate shift* and proposed a solution called Batch Normalization [16].

Their proposed solution is to normalize the inputs of internal layers, allowing to use much higher learning rates and be less careful about initialization [16]. The mean and variance needed to normalize the inputs of a hidden layer are determined for each batch separately. Because at test time batches are no longer available, as the trained model predicts on single input data, a moving

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.14.: Batch Normalizing Transform, applied to activation x over a mini-batch [16]

.

average of the mean and variance values is computed during training and then used after training to approximate the values for normalization [10].

2.2.4.7. Model Architecture

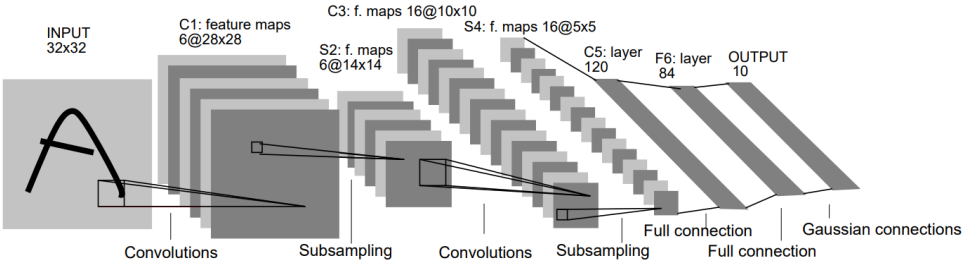


Figure 2.15.: LeNet 5 CNN Architecture [20]

In standard Convolutional Neural Network (CNN) architectures the first part of the model consists of a series of convolutional layers, activation functions and pooling layers. The combination of these layers is stacked multiple times, decreasing the height and width of the input, but increasing the depth by adding feature maps. While starting to recognize low-level features like edges, the deeper the network is, the more abstract the extracted features get. If the input is reduced enough,

the cubic volume of width*height*depth(number of feature-maps) gets flattened and the second part of the architecture is then a traditional fully connected neural network which learns the final abstractions needed for the dedicated task [20].

While this architecture performs well on a variety of different tasks, during the last decade, numerous improvements have been proposed and some of them inspired the architecture used in this project:

- Convolution and Pooling layers have been stacked until the input was reduced from 480*640*1 to 7*10*128. 128 come from the chosen amount of filters and has been chosen arbitrarily. The variation of this hyper-parameter is discussed in section 3.2.

- The activation function "ELU" [4] was chosen in combination with a favourable parameter initialization strategy - the He Normal Initialisation [13].

- Zero Padding was chosen to let the Average Pooling layers be the only layers reducing the width and height of their inputs.

- Batch Normalization was performed before every Pooling Layer to increase robustness against internal covariate shift [16].

- Only the 32 filters inside the first convolutional layer have a size of 7*7, the rest of the convolutions use filter sizes of 3*3 and 1*1. These size settings were first used by GoogLeNet, which won the ILSVRC challenge 2014 [36]. The 1*1 filters seem to miss the purpose of convolutions, which is to learn spatial features and reduce the dimensionality of inputs [20]. However, it is important to remember the convolutions third summation along the depth of the input of the convolution from Equation 2.5 which is responsible for the convolutional layers ability to also learn depth-related features. By using fewer filters than feature maps in the previous layer, a 1*1 filter also reduces the depth of the model without losing depth-related information, thus reducing the total amount of model parameters. So having one 1*1 Filter before a 3*3 filter can be interpreted as one single filter with enhanced depth awareness [10].

```
model = keras.models.Sequential([
    keras.layers.Conv2D(32, 7, activation="elu", padding="same", kernel_initializer="he_normal",
                        input_shape=[TARGET_HEIGHT,TARGET_WIDTH,CHANNELS]),
    keras.layers.Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(16, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(16, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(32, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(64, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(32, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(64, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(64, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(128, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.AveragePooling2D(pool_size=2),
    keras.layers.Conv2D(64, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.Conv2D(128, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.GlobalAvgPool2D(),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation="elu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(16, activation="elu"),
    keras.layers.Dense(1, activation="relu")
])
```

Figure 2.16.: Model Definition using Keras Sequential API

- Global Average Pooling was used as a final pooling layer to further reduce the dimensionality of the data from 7*10*128 to 1*1*128. This may seem to destruct spacial information, but as this point after all the pooling layers and continuous abstraction steps, there is no more critical spacial information left. Furthermore, Global Average Pooling helps to prevent overfitting [10].

- The last part of the architecture consists of small densely connected neural network with 20% dropout applied to the second layer. The numbers of this neural network again are chosen arbitrarily and variations of this hyper-parameter is discussed in section 3.2.

- The final output of the network needs to be a single neuron with a linear activation function for the given regression problem. ”ReLU” has been chosen as an activation to ensure that

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 480, 640, 32)	1600
conv2d_1 (Conv2D)	(None, 480, 640, 32)	9248
batch_normalization (BatchNo	(None, 480, 640, 32)	128
average_pooling2d (AveragePo	(None, 240, 320, 32)	0
conv2d_2 (Conv2D)	(None, 240, 320, 16)	528
conv2d_3 (Conv2D)	(None, 240, 320, 32)	4640
batch_normalization_1 (Batch	(None, 240, 320, 32)	128
average_pooling2d_1 (Average	(None, 120, 160, 32)	0
conv2d_4 (Conv2D)	(None, 120, 160, 16)	528
conv2d_5 (Conv2D)	(None, 120, 160, 32)	4640
batch_normalization_2 (Batch	(None, 120, 160, 32)	128
average_pooling2d_2 (Average	(None, 60, 80, 32)	0
conv2d_6 (Conv2D)	(None, 60, 80, 32)	1056
conv2d_7 (Conv2D)	(None, 60, 80, 64)	18496
batch_normalization_3 (Batch	(None, 60, 80, 64)	256
average_pooling2d_3 (Average	(None, 30, 40, 64)	0
conv2d_8 (Conv2D)	(None, 30, 40, 32)	2080

(a)

conv2d_9 (Conv2D)	(None, 30, 40, 64)	18496
batch_normalization_4 (Batch	(None, 30, 40, 64)	256
average_pooling2d_4 (Average	(None, 15, 20, 64)	0
conv2d_10 (Conv2D)	(None, 15, 20, 64)	4160
conv2d_11 (Conv2D)	(None, 15, 20, 128)	73856
batch_normalization_5 (Batch	(None, 15, 20, 128)	512
average_pooling2d_5 (Average	(None, 7, 10, 128)	0
conv2d_12 (Conv2D)	(None, 7, 10, 64)	8256
conv2d_13 (Conv2D)	(None, 7, 10, 128)	73856
batch_normalization_6 (Batch	(None, 7, 10, 128)	512
global_average_pooling2d (Gl	(None, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 32)	4128
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

Total params: 228,033
Trainable params: 227,073
Non-trainable params: 960

(b)

Figure 2.17.: Model Summary with Layer Dimensions

predictions are only positive, but a simple linear output $f(x) = x$ would be a viable choice too.

2.2.5. Loss Function

The combination of output activation function, data labels and the used loss function together frame the behaviour of a model. The amount of possible combinations gives Neural Networks great flexibility, thus allowing a wide range of different application like Classification, Regression, Segmentation etc.

In the case of this project, the expected prediction will be a single continuous value, describing the angle of the motor shaft to a reference point in the range of 0 - 2π. This is a regression problem requiring our model to have a single neuron as an output with a linear activation function. The

default loss function for such a task is the Mean Absolute Error (MAE) and its close relative, the Mean Squared Error (MSE) [32].

This would work to a certain extent but is not optimal at the extreme values for periodic problems. Predicting the value 6.0 ($2\pi - 0.283$) for the real value of 0.0 would be a good prediction, considering the angle between the prediction and the ground truth is only about 0.283 radians. However, using MAE as a loss would penalize the model for being off by 6.0 radians. This little investigation shows, that a loss function is needed, which also considers the periodic nature of rotational systems. For the given reasons a loss function consisting of both, the MSE and the squared angular difference was chosen.

$$loss = \frac{1}{n} * (\arctan(\frac{\sin(y - \hat{y})}{\cos(y - \hat{y})})^2 + (y - \hat{y})^2) \tag{2.6}$$

2.2.6. Adaptive Momentum Estimation Optimization

Optimization Algorithms are of one of the main tools of many engineers and build the foundation of any ML project. While simple algorithms like Stochastic Gradient Descent have been responsible for many successes (Deng et al., 2013; Krizhevsky et al., 2012; Hinton & Salakhutdinov, 2006; Hinton et al., 2012a; Graves et al., 2013) Stochastic Optimization is still a very active research field which yielded many improved algorithms over the last decade.

One of these advanced optimization algorithms is Adaptive Momentum (ADAM) [18]. Requiring only first-order gradients, ADAM is efficient and has lower memory requirements than second or third-order methods. ADAM combines the advantages of AdaGrad [30] and RMSProp by having individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [18]. To put it in simple words, instead of using only momentary gradient information at each training step like SGD does, ADAM models *momentum* of the gradients by scaling the learning rate with the Exponentially Weighted Moving Averages ($m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$) forming the moment estimations [18]. The momentum part of the

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 2.18.: Adaptive Momentum Estimation Optimization Algorithm [18]

algorithm hinders fast changes to the learning rate, thus being able to deal with very noisy and sparse gradients and non-stationary objectives and problems [18].

Fortunately, TensorFlow provides an implementation of ADAM with very good default parameter setting, requiring very little hyper-parameter tuning to make ADAM work in a variety of different applications.

2.2.7. Training Settings

The training was conducted over 45 epochs, which are complete runs over the entire training set. With 80.000 training images, this corresponds to 3.600.000 single training steps in which $1,16 * 10^{12}$ individual pixels are processed.

A batch size of 16 was used because a higher number would have required the used working station to have more RAM available to store all the gradients and activation caches of every image inside a batch.

Adaptive Momentum (ADAM) Optimization was used to minimize the models custom loss (see section 2.2.5) over the augmented training examples.

Instead of using a constant learning rate, which is a very sensitive hyper-parameter and if chosen uncarefully can easily slow down convergence drastically or make the model diverge during training, Learning Rate Scheduling was used to leverage both advantages of high and low learning rates:

- high learning rates which will still converge, do so faster than lower learning rates but tend to oscillate over the global loss minimum, which leads to worse accuracies [38].
- lower learning rates converge slower, but the found solution is more accurate than the ones found by higher learning rates [38].

A stepwise scheduler was used which started with a learning rate of 0.004, which is four times higher than the one suggested in the original ADAM paper [18], and stopped with a learning rate of 0.0005 at epoch 30.

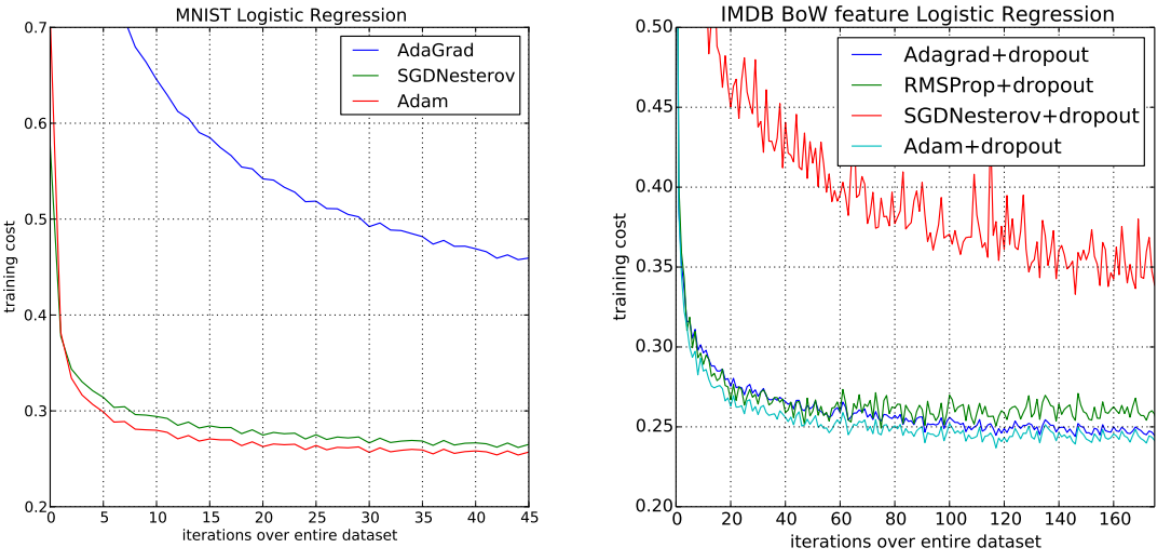


Figure 2.19.: Empirical Comparison of Different Optimization Algorithms [18]

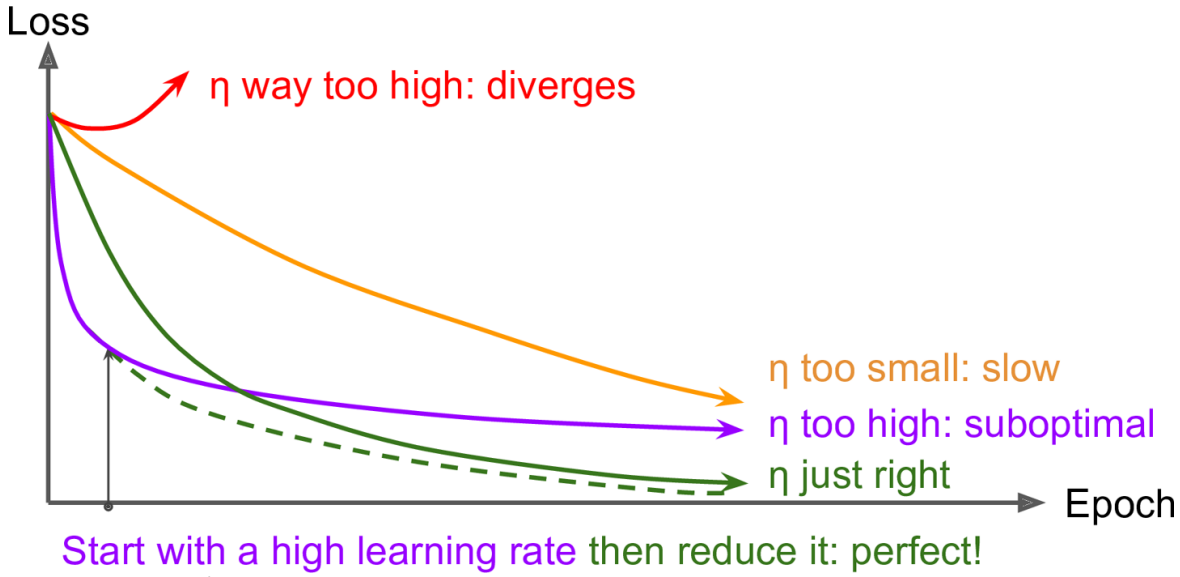


Figure 2.20.: Learning Rate Scheduling with $\eta = learningrate$ [10]

3. Results

3.1. Training Results

The training run over 45 epochs on 80 thousand training images ran for about 1 day and 16.5 hours on the hardware described in section 2.2.1. Table 3.1 shows the evaluation of the trained model on the 3 different data sets and Fig. 3.1 depicts the training curve of the MAE on both the training set (orange) and the validation set (blue).

The fast progress during the first epochs slowed down at around epoch 8 and eventually converged to an MAE of 0.06 radians, which is better than expected and clear evidence that the trained model has low bias. With such a low MAE one would expect the model has over-fitted the data (high variance), but the small difference between the training and validation curves proofs that the model was able to generalise to new images it had not seen during training.

One should, however, take a more sober look at the validation and test set. Due to former scepticism, the variations of both sets to the training set are modest. Live tests of the model under more difficult lighting conditions and with different servo motor models showed an MAE of approximately 0.1 - 0.2, which is still a very good result and proof of the model's capability to generalise. A greater difference between the training and validation sets would also allow tuning the network towards greater performance without generalisation compromises, which turned out to be a challenge with both training and validation curves being so close together.

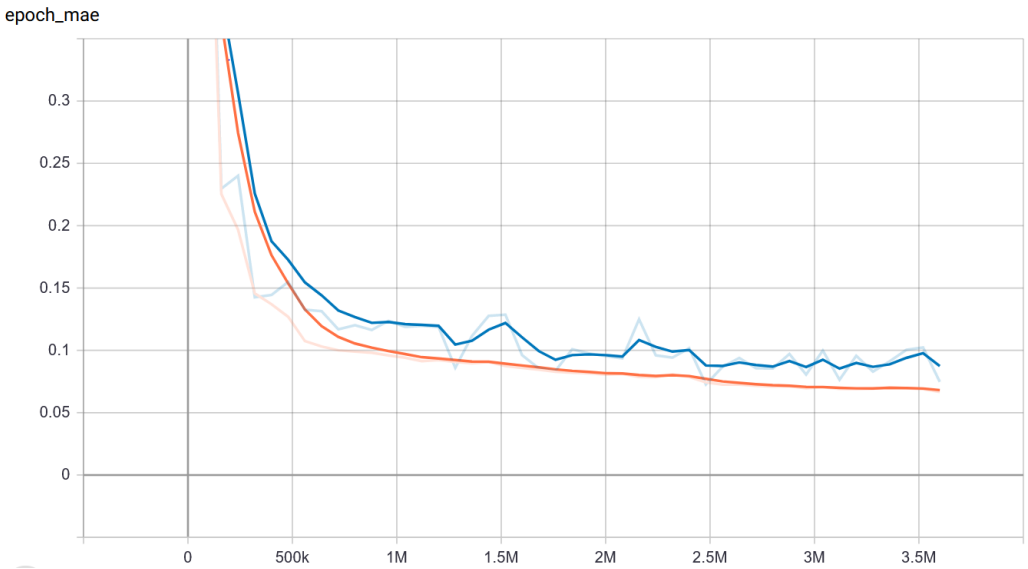


Figure 3.1.: Training Curve: MAE

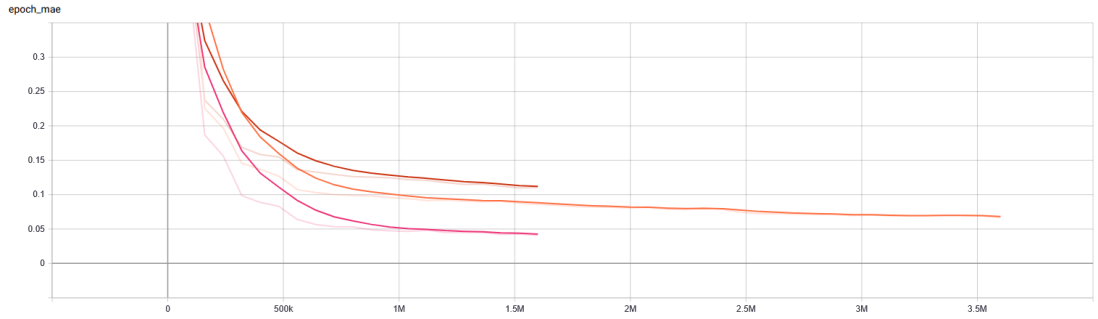
Table 3.1.: Model Evaluation

metrics	train set	validation set	test set
custom loss (2.6)	0.0554	0.1055	0.0625
mean absolute error	0.0666	0.0747	0.0638

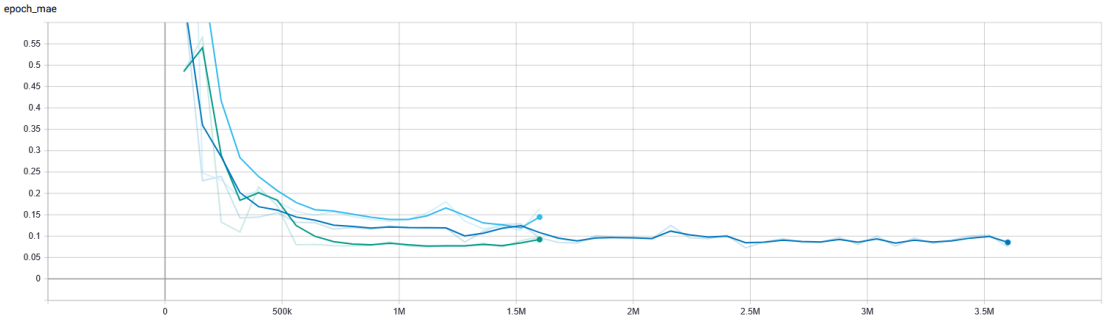
3.2. Model Size Variations

To examine the effect of the model size and the number of filters and neurons chosen for the convolutional and densely connected layers, two additional training runs were conducted with one smaller and one bigger model over 20 epochs.

The smaller model (red curve in 3.2a and light blue curve in 3.2b) had the exact same architecture as the one described in section 2.2.4.7, but with the number of filters quartered and the number of neurons halved, it is roughly 15 times smaller than the original one. Considering the big difference in size one would expect the smaller model to perform considerably worse and having more bias than the original model (see section 2.1.3). Surprisingly though, the small model had only a MAE of 0.1445 radians (8.27°) on the test set, which is roughly two times the MAE of the original model. With only 15 thousand parameters, the training of the model needed considerably fewer hardware resources and converged faster during the first epochs.



(a) Training MAE



(b) Validation MAE

Figure 3.2.: MAE Training Curves of Three Differently Sized Models

In the bigger model (pink curve in 3.2a and green curve in 3.2b) the amount of neurons in the densely connected layers were increased from 32 and 16 to 256 and 256. In addition to this change, the dropout layer between these two densely connected layers was removed to further examine the generalization capabilities of this model. Interestingly, this was enough to let the bigger model have a higher variance than the original model. Due to its higher numbers of neurons in the last layers, it was able to learn more patterns in the training set, which led to a smaller MAE on the training set than the original one had. But the MAE of 0.1445 radians on the test set is the same as the one of the smaller model, which is a clear indication, that the features this model learned did not generalize well on new data.

Table 3.2.: Model Size Comparison - MAE

number of model parameters	train set	validation set	test set
15.425	0.1105	0.1641	0.1445
228.033	0.0666	0.0747	0.0638
322.433	0.0408	0.0981	0.1445

3.3. Data Set Size Variations

One very interesting question related to every ML project is how big the training set needs to be. Due to the common notion among ML practitioners, that there can not be too much data and because of a general sparsity of freely available, big and high-quality data sets, the acquisition and labelling of training data is an essential task, which can quickly consume a lot of project resources.

According to a paper published by J. Hestness et. al. in 2017 [14], the training data set size can be qualitatively classified by three different regions.

The region in which the error of the model will not decrease further is called "irreducible error region". This error is caused by noise, data labelling errors or systematic errors in the data acquisition process [14]. In the context of this work, one part of this error is caused by the 12 bit rotary encoders used to label the training data. Taking this maximum precision into consideration, a minimal theoretical MAE of $\frac{2*\pi}{2*2^{12}} = 0.00077$ can be assumed. In practice, the optical limitations of the used camera [26] will increase this value.

Inside the small data region, the model can only guess when predicting, as there is simply not enough training data to learn the input-output relationship [14].

After the small data region, the power-law region follows, where each new training sample helps the model to generalise better to data it had not seen during training. The improvement in this region decreases exponentially, which means that after a certain threshold, adding new training samples will not reduce the training error noticeably. At this point any further data acquisition efforts, especially if resource-intensive acquisition methods are used, should be carefully considered [14].

Being in a position where automated data acquisition grants cheap and fast access to a big data set with predefined quality standards [9], allows a "brute force" approach, where the training set size is determined without worrying too much about wasting resources for acquisition. However, having a very big data set does come at the cost of increasing difficulty of handling all the data

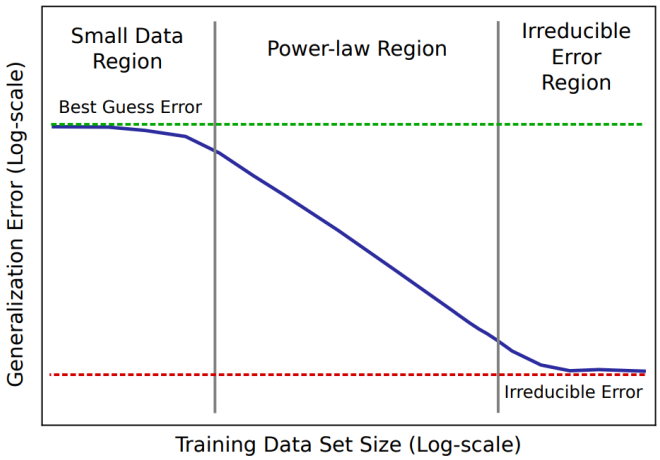


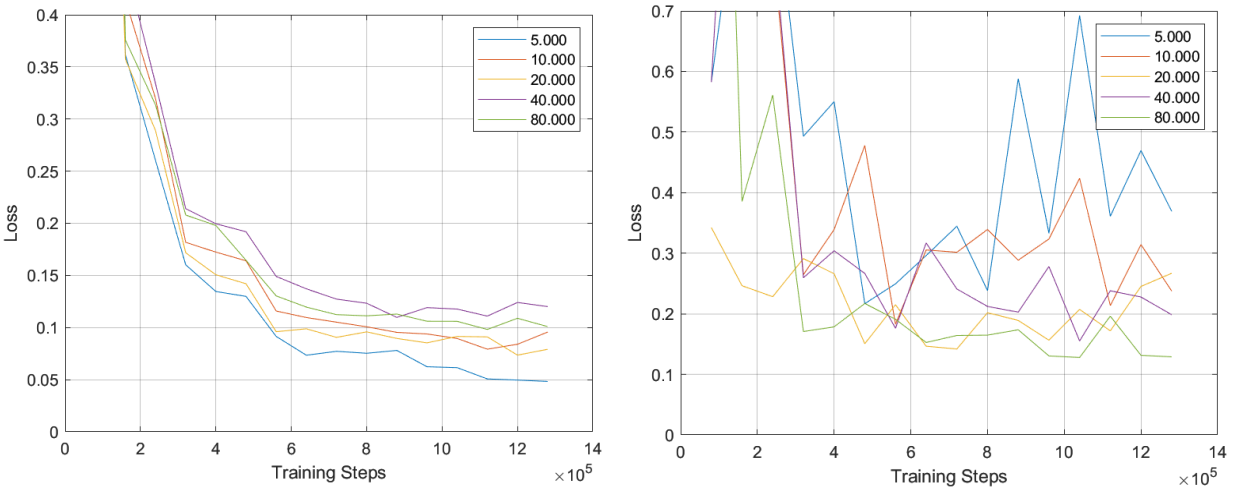
Figure 3.3.: Power-Law Learning Curves [14]

and increased memory requirements. By using the data set size uncritical DataFrame format (see section 2.2.2) the increased complexity of the handling of a large data set has been taken care of. Nonetheless, four training runs with the same model presented in section 2.2.4.7, but with different proportions of the same data set [40.000, 20.000, 10.000, 5.000] has been conducted over 1.280.000 training steps to examine how different training set sizes impact the performance on the given problem.

Table 3.3.: Data Set Size Variations

Data Set Size	Training MAE	Validation MAE
80.000	0.0907	0.0860
40.000	0.1021	0.1814
20.000	0.0843	0.1963
10.000	0.0853	0.1551
5.000	0.0741	0.1876

All models were able to fit the training data reasonably well with a tendency to fit the training data better when trained on a smaller data set but perform worse on the validation data. The augmentations that vary each validation are responsible for the more agitated and noisy validation curves in Figure 3.4b. Over-fitting due to a lack of training data can be observed in validation curves when after a first decline, the loss starts to increase again. It is at this point when the model stops learning features that help him generalise and starts to learn the training data ”by heart”. This behaviour is only noticeable in the models trained on 20.000, 10.000 and 5.000 examples.



(a) Training Curves

(b) Validation Curves

Figure 3.4.: Data Set Size Variations

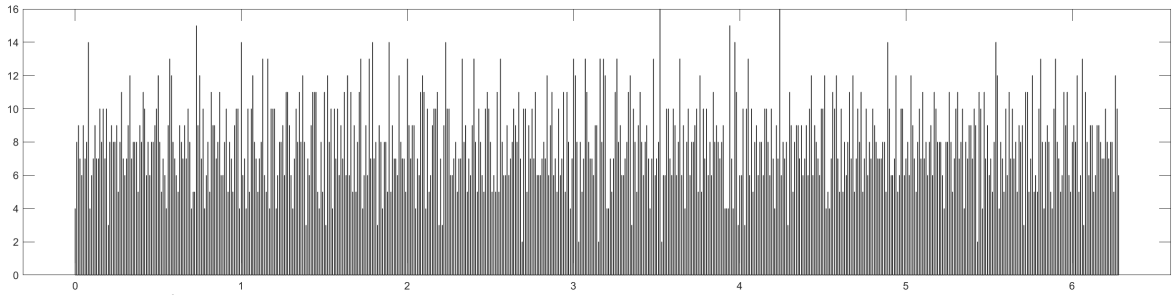


Figure 3.5.: Histogram of the first 5.000 labels with a count of 4.096 bins

The degree to which over-fitting is accepted certainly depends on the solution requirements, but due to the milder validation set conditions, training sets over 40.000 samples should definitively be prefered.

An interesting result is that with a validation MAE of 0.1876 radians (10,75°), the model trained on 5.000 training samples still is able to predict the angle of the motor shaft with reasonable accuracy. The 4.096 encoder steps in conjunction with the random sampling method used during the data acquisition [9] leads to some possible values not being represented. In classification problems, a class being underrepresented to such an extent would lead to very poor performance [10]. However, in regression problems, the possible ”classes” are very similar, which combined with the used data augmentation enables the trained model to ”interpolate” its learned knowledge to values not present in the training set.

3.4. Statistical Analysis of the Vision Sensor

The following analysis is based on the predictions of the model, for which the training results have been presented in section 3.1, on the 10.000 unaugmented test images from the test set.

The purpose of this analysis is to gain some insights at the performance and behaviour of the model beyond the one obtained by the training loss and metrics. These results are especially interesting because the loss and metrics used so far are mean values over the entire data sets, which does not say anything about the distributions and deviations of the predictions. The huge advantage of having a single continuous output allows for a variety of easy to interpret statistical plots, which would not be as easy and interpretable when dealing with classification problems.

Insights gained from this analysis can be used to evaluate the trained sensor and design filters to deal with statistical outliers, thus making the sensor system more suitable for accuracy critical tasks.

Table 3.4.: Test Set Performance Measures

metric	Labels	Predictions	Angular Difference
Mean	3.1386	3.1404	-0.0081
Variance	3.2787	3.2298	0.0134
Standard Deviation	1.8107	1.7972	0.1157

Examining the statistical results yielded the following observations:

- The mean, variance and standard deviation of the test labels, the predictions and their angular difference do not show any peculiarities.

- The Boxplots of both the label and the prediction distribution (Figure 3.6a) agree with the previous point in that they do not vary significantly. What is noticeable though is that the prediction distribution does not have any outliers smaller than 0 or bigger than 2π.

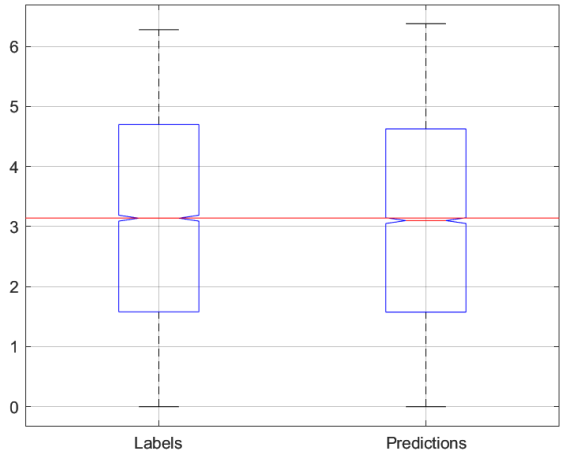
- Unsurprisingly the Boxplot of the angular difference distribution between labels and predictions does include outliers ranging from $-\pi$ to π. This means that for some images

the sensor predicts completely unusable results, which should definitively be considered when building safety-critical systems.

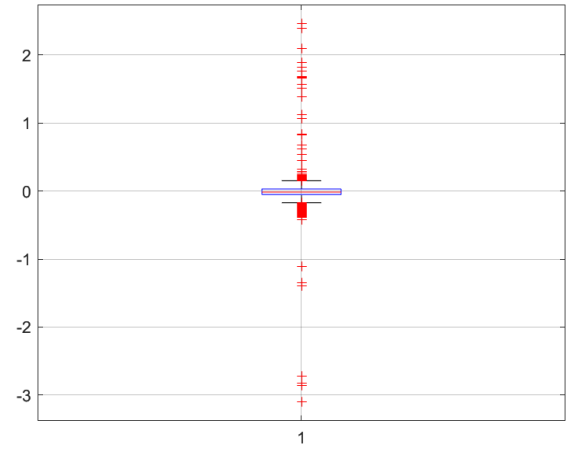
- The histogram of the angular differences (Figure 3.6c) is the first indication, that there is some kind of systematic error, as the subjective centre of this distribution is negative, which means that the sensor tends to predict values that are slightly bigger than their "ground truth" values. Another observation is that he distribution with its faster incline for negative values and slower decline of positive values is not a Gaussian distribution which is confirmed by the probability plot of the angular difference (Figure 3.6d) which using this scale for the y-axis should be linear for normal distributions.

- Another deviation from optimal behaviour closely related to the non-gaussian distribution of the angular differences in the smaller probability of predictions around the points $[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}]$ evident in the polar histogram of the predictions (Figure 3.6f). What is even more interesting is that while there are fewer predictions at these points, the prediction accuracy is higher, as the scatter plot of the angular differences between the labels and the predictions plotted against the corresponding angle values (Figure 3.7) demonstrates. Looking at the front panel of the servo motor (Figure 3.8) this could be explained with the overlapping of the motor shaft with noticeable optical features like the line indicating position 0, the gears at position $\frac{\pi}{2}$ and $\frac{3\pi}{2}$, or the screw and encoder cable at position π. While this intuition may contribute to this phenomenon, a thorough investigation would be very interesting and might deliver some deeper insights into the inner workings of the used model architecture.

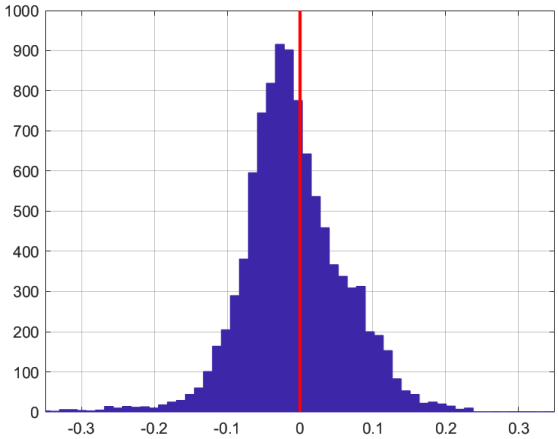
- Other peculiarities of the scatter plot of the angular differences between the labels and the predictions plotted against the corresponding angle values (Figure 3.7) include a suspiciously linear increase at the start of the plot, a slightly periodical behaviour for every quadrant and most outliers being at the extreme points and around the value 2 where there is a lot of text on the front panel of the motor.



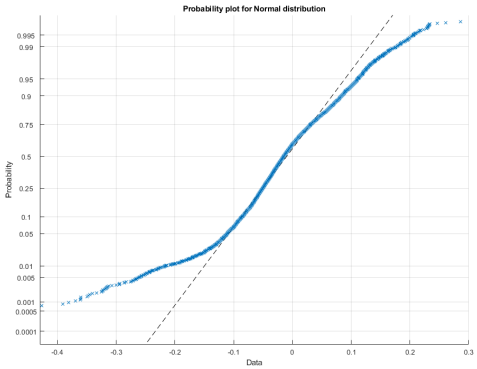
(a) Label and Prediction Distribution



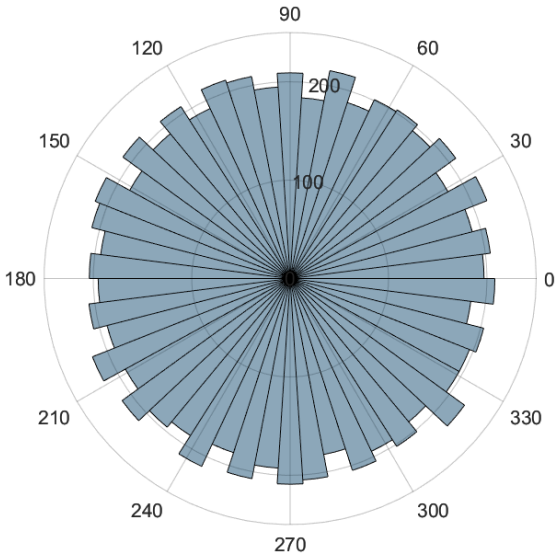
(b) Angular Difference Boxplot



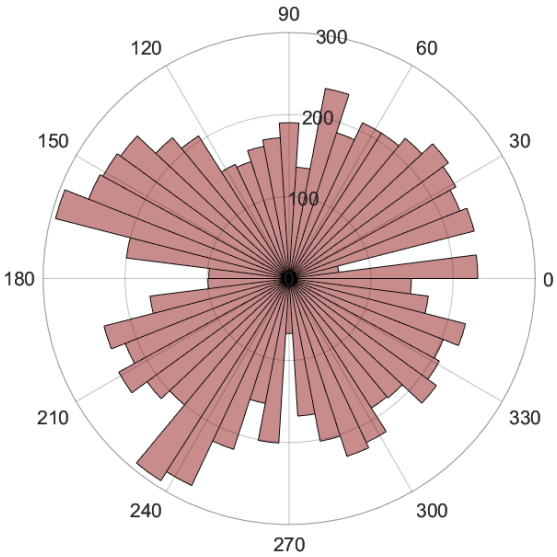
(c) Angular Difference Histogram



(d) Probability plot for angular difference



(e) Polar Histogram of Labels



(f) Polar Histogram of Predictions

Figure 3.6.: Statistical Analysis of Test Set Labels, Predictions and their Angular Difference

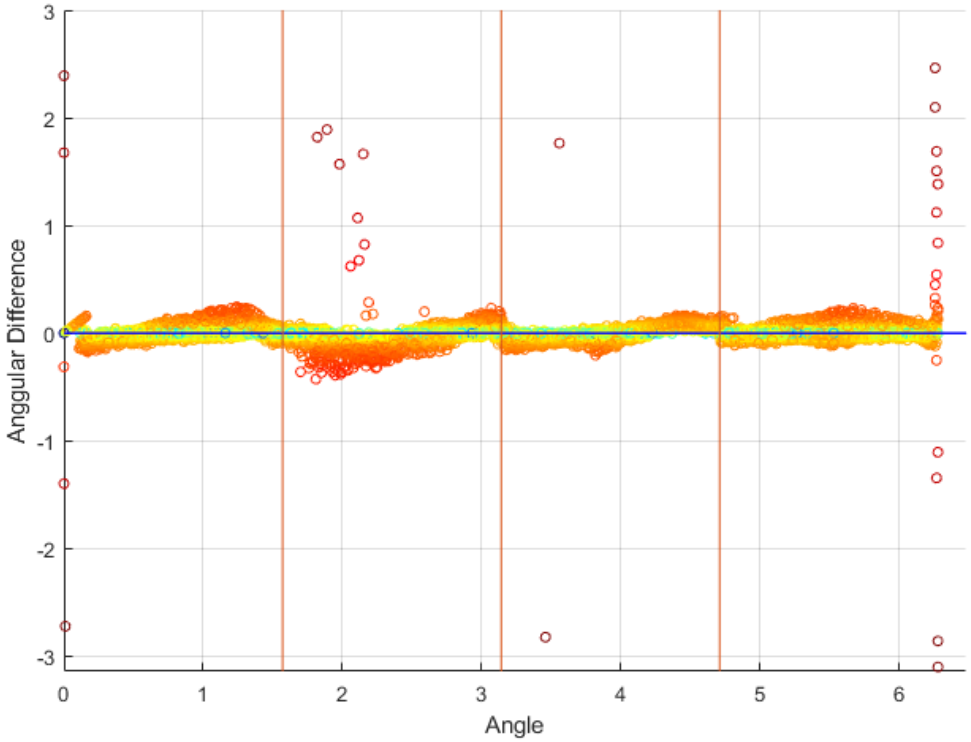


Figure 3.7.: Scatter Plot of Angular Difference against Angle Values

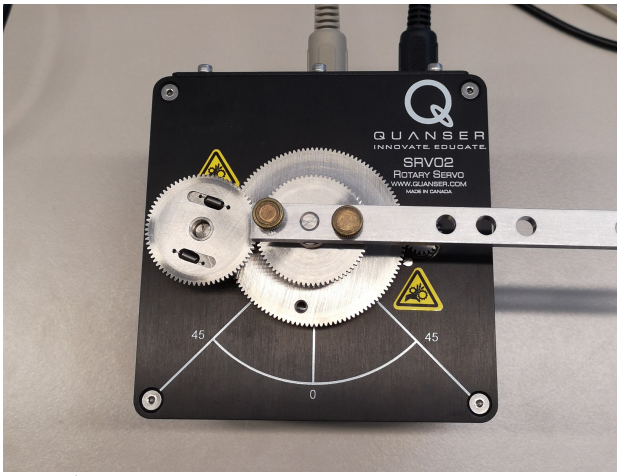


Figure 3.8.: Servo Motor Front Panel [27]

4. Discussion

4.1. Method Justification

Justifying the chosen labour and resource-intensive approach to solve a sensor problem which can be solved trivially with a cheap, fast and accurate motor encoder was inherently difficult throughout the entire project. One reason for this is that the focus of this work has never been on directly applicable or spectacular results, but rather on exploring a fascinating method and applying it on resources available at the CUAS on a problem related to the rest of our studies.

However, I want to use this opportunity to defend the chosen approach with a few remarks:

- There are old machines and measurement equipment still in use, that do not have the degree of connectivity and accessibility of information found in modern equipment. These devices depend on humans evaluating their (often analogue) displays and therefore hinder the companies ability to increase their level of automation. A Vision Regression Sensor very similar to the one proposed in this work could be used to replace the human intervention and achieve higher automation even when working with devices not designed for interoperability.

- The decision to use a fast controllable actor with the possibility to accurately measure the value, that we want our ML based sensor to predict allowed for a high degree of automation of the data acquisition process, which in turn ultimately lead to the success of the training (see section 3.3).

- The general nature of the utilized ML algorithms allows their use in a variety of different applications and even different domains. The data set has to be replaced and probably the

model has to be adjusted to a certain degree, but the insights gained during this work can be of great value for other ML projects.

4.2. Data Set Size and Quality

The results from section 3.3 demonstrated the big impact of the data set size on the performance and generalization capabilities of the trained model. Reducing the number of training samples to 40.000 almost doubled the validation error and reducing the number of training samples to 20.000 yielded even worse results. Personally, I had not expected the training set size to have that big of an impact, as I expected the model to not even have to be trained on every possible output value in order to interpolate between two "learned" points. The fact that a lot of careful planning went into the data augmentation strategy deepened my original expectations that around 10.000 training samples would suffice in learning to predict all 4096 encoder steps properly. But it turned out, that for the given problem, 80.000 training samples is not an unreasonable number of samples for a training set.

This consents with my experience that most other application-focused ML projects I came in touch with struggled because of a lack of available training data with sufficient quality. That is why I think that in the short to mid-term, most promising research tasks include those that reduce the amount of needed training data, including transfer learning [37], unsupervised pre-training [6] or capable general intelligence approaches that are able to learn from smaller amounts of data because of their general understanding at a given task [11].

In hindsight, I regret not making the test and validation set more diverse from the training set. Former scepticism during the project planning phase and a lack of ML experienced led to a more careful validation and test set design, but tests with the final model under poor lighting conditions, with hands covering important parts of the motor, or even with other servo motor models with different front plates and other motor shafts showed promising results and demonstrated the fascinating generalization capabilities of a properly trained CNN. Having such a difficult test set with harsher variations from the training set conditions would have yielded worse validation and

test results and less impressive metrics, but would have allowed for a better evaluation of the generalization capabilities of the model and a proper hyper-parameter tuning during training.

4.3. Required Knowledge

With Machine Learning being such a broad and dynamic field with roughly 50 to 100 research papers published daily on arxiv.org only [1], one may find the overwhelming amount of information and burden of entry scary and too much to cope with. But being such a mature and sought after research subject, a lot of the tooling and educational material has matured too, allowing for a careful and approachable entry to this field. With high-level frameworks like Keras [2], even without much detailed knowledge about the intricate implementation details of the ML algorithms working in the background, one might get pretty far on an application-level by just applying some of the well documented architectures and train them on new data. I am not saying that one does not need to dedicate a lot of time to build up a thorough understanding of the topic but during my first smaller ML projects I found the tooling and material available to be especially accessible which lets me hope that more engineers will adopt these tools and techniques into their toolbox and that in the next years we will see more projects not focused on the development of sophisticated model architectures but applying the well-studied ones to many problems which could greatly benefit from ML advantages.

What in my opinion is useful and even required if one wants to optimise their own ML projects is a certain degree of proficiency in handling data in many sizes and formats. Due to the complex nature of ML algorithms and the many layers of computations during training, it can often be difficult to identify errors or shortcomings of the trained model and finding the origin of these errors often does require a deeper understanding of the math behind the used algorithms. Personally, I find that these unexpected outcomes and the involved investigation of the inner workings of the algorithms to be especially appealing and one of the points that make working on ML projects so intriguing and compelling.

5. Conclusion

This thesis has aimed to explore Deep Learning based solutions and apply them to a Computer Vision problem. Predicting the angular position of a servo motor shaft from images has been chosen as a measurement problem, because it seemingly fits in the mechatronics based education at the Carinthia University of Applied Sciences and offers access to a large data set through the high degree of automation achieved in the data acquisition discussed in [9].

Chapter 1 and 2 provided an introduction to many Machine Learning concepts followed by a more in-depth view of the building blocks used in the Convolutional Neural Network architecture designed for the vision regression sensor proposed in this work.

The training results discussed in chapter 3 surprised with their high prediction accuracy with a Mean Absolute Error of only 0.0638 radians (3,66°) and very interesting prediction distributions discovered during the statistical analysis. The rest of the 3^{rd} chapter presents the results of training runs conducted with variations in the model and data set size and demonstrated that once there is enough training data, the details of the chosen architecture are not as important as originally expected.

I am really happy with the outcome of this project. I was very happy when I got the opportunity to work on a project of my choice, but I had not expected the approach to work that well on the chosen problem.

However, there are still a few things I would like to improve:

- The difference between the validation and training data sets needs to be increased to further optimize the training for higher generalization capabilities.

- I expect that including some kind of feedback or time awareness to the model architecture would greatly increase its reliability. This could be done by designing some kind of filter for the output of the sensor or choosing a sequential model architecture [10], which would require the reorganization of the data acquisition, to also include time-stamps to the acquired labels.
- In hindsight, choosing "ReLU" as the final output activation function to ensure that the predictions stay positive was a poor choice and might be responsible for the behaviour of the predictions around very small values.

I am very optimistic about the future of Machine Learning based research, both on the application and theoretical level. I think that for ML-based algorithms to find more use in a broader field of applications and industries, there needs to be more insight into the decision process of trained ML models. I think that model architecture that can cope with smaller data set sizes will also empower a lot of applications that struggle to utilize Deep Learning because of a lack of data.

6. Bibliography

[1] *arXiv.org*. [Online]. Available: https://arxiv.org/list/cs.LG/pastweek?skip=132&show=25.

[2] Chollet, F., *keras*, https://github.com/fchollet/keras, 2015.

[3] Cislo, P., *Pin by Pawel Cislo on Data Science in 2019: Machine learning artificial intelligence, Ai machine learning, Computer coding*, Nov. 2019. [Online]. Available: https://www.pinterest.at/pin/672232681855858689/.

[4] Clevert, D.-A., Unterthiner, T., and Hochreiter, S., "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", Jan. 2016.

[5] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, cite arxiv:1810.04805Comment: 13 pages, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805.

[6] Dorogyy, Y. and Kolisnichenko, V., "Unsupervised Pre-Training with Spiking Neural Networks in Semi-Supervised Learning", *2018 IEEE First International Conference on System Analysis Intelligent Computing (SAIC)*, 2018. DOI: 10.1109/saic.2018.8516733.

[7] Elloumi, M., Granitzer, M., Hameurlain, A., Seifert, C., Stein, B., Tjoa, A. M., and Wagner, R., *Database and Expert Systems Applications: dexa 2018 international workshops*. Springer, 2018.

[8] *Franka Emika Robot*. [Online]. Available: https://www.franka.de/.

[9] Fritz, G., *Vision Control I - Automated Data Acquisition for Supervised Learning*, Jan. 2020.

[10] Géron, A., *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017, ISBN: 978-1491962299.

[11] Goertzel, B. and Pennachin, C., *Artificial General Intelligence*. Springer-Verlag Berlin Heidelberg, 2007.

[12] Goodfellow, I., Bengio, Y., and Courville, A., *Deep learning*. The MIT Press, 2017.

[13] He, K., Zhang, X., Ren, S., and Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", *CoRR*, vol. abs/1502.01852, 2015. arXiv: 1502.01852. [Online]. Available: http://arxiv.org/abs/1502.01852.

[14] Hestness, J., Narang, S., Ardalani, N., Diamos, G. F., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y., "Deep Learning Scaling is Predictable, Empirically", *CoRR*, vol. abs/1712.00409, 2017. arXiv: 1712.00409. [Online]. Available: http://arxiv.org/abs/1712.00409.

[15] HutsonMay, M., MervisDec, J., WadmanDec, M., MlotDec, C., GrimmDec, D., and FrederickDec, E., *AI researchers allege that machine learning is alchemy*, May 2018. [Online]. Available: https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy.

[16] Ioffe, S. and Szegedy, C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.

[17] Kim, P., *MATLAB Deep Learning: with Machine Learning, Neural Networks and Artificial Intelligence*. Apress, 2017.

[18] Kingma, D. P. and Ba, J., "Adam: A Method for Stochastic Optimization", *CoRR*, vol. abs/1412.6980, 2014.

[19] Kumarsingh, B., Verma, K., and Thoke, A. S., "Investigations on Impact of Feature Normalization Techniques on Classifier's Performance in Breast Tumor Classification", *International Journal of Computer Applications*, vol. 116, no. 19, pp. 11–15, 2015. DOI: 10.5120/20443-2793.

[20] Lecun, Y., Haffner, P., Bottou, L., and Bengio, Y., "Object Recognition with Gradient-Based Learning", *Shape, Contour and Grouping in Computer Vision Lecture Notes in Computer Science*, pp. 319–345, 1999. DOI: 10.1007/3-540-46805-6_19.

[21] Marsland, S., *Machine Learning An Algorithmic Perspective*, 2nd ed. Chapman Hall/CRC, 2014.

[22] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[23] Mikołajczyk, A. and Grochowski, M., "Data augmentation for improving deep learning in image classification problem", in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, May 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[24] Ng, A., *Machine Learning | Coursera*, 2019. [Online]. Available: https://www.coursera.org/learn/machine-learning.

[25] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S., "Activation Functions: Comparison of trends in Practice and Research for Deep Learning", *CoRR*, vol. abs/1811.03378, 2018. arXiv: 1811.03378. [Online]. Available: http://arxiv.org/abs/1811.03378.

[26] *PlayStation Eye Camera*. [Online]. Available: https://en.wikipedia.org/wiki/PlayStation_Eye.

[27] *Quanser*. [Online]. Available: https://www.quanser.com/.

[28] Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. DOI: 10.1037/h0042519.

[29] Rossum, G. van, *Python 3.7.6 documentation*. [Online]. Available: `https://docs.python.org/3.7/`.

[30] Ruder, S., "An overview of gradient descent optimization algorithms", *CoRR*, vol. abs/1609.04747, 2016. arXiv: `1609.04747`. [Online]. Available: `http://arxiv.org/abs/1609.04747`.

[31] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation", Jan. 1985. DOI: `10.21236/ada164453`.

[32] Russell, S. and Norvig, P., *Artificial intelligence A Modern Approach*, 3rd ed. Pearson Education, 2010.

[33] Santos, L. A. d., *Pooling Layer*. [Online]. Available: `https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pooling_layer.html`.

[34] Scherer, D., Müller, A., and Behnke, S., "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition", *Artificial Neural Networks – ICANN 2010 Lecture Notes in Computer Science*, pp. 92–101, 2010. DOI: `10.1007/978-3-642-15825-4_10`.

[35] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: `http://jmlr.org/papers/v15/srivastava14a.html`.

[36] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., "Going deeper with convolutions", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. DOI: `10.1109/cvpr.2015.7298594`.

[37] Wong, C., Houlsby, N., Lu, Y., and Gesmundo, A., "Transfer Automatic Machine Learning", *CoRR*, vol. abs/1803.02780, 2018. arXiv: `1803.02780`. [Online]. Available: `http://arxiv.org/abs/1803.02780`.

[38] Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., and Zhang, Q., "Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks", *arXiv e-prints*, arXiv:1908.06477, arXiv:1908.06477, Aug. 2019. arXiv: `1908.06477 [cs.LG]`.

[39] *Zotac Gaming GeForce RTX 2080 Ti Twin Fan*, Aug. 2019. [Online]. Available: https://www.zotac.com/de/product/graphics_card/zotac-gaming-geforce-rtx-2080-ti-twin-fan.

A. Appendix - Glossary

Acronyms

ADAM Adaptive Momentum. 10, 30–32

AI Artificial Intelligence. 11

ANN Artificial Neural Network. 8

API Application Programming Interface. 12, 14

CNN Convolutional Neural Network. 4, 20, 22, 23, 26, 46

CSV Comma-Seperated Values. 13

CUAS Carinthia University of Applied Sciences. 1, 45, 49

MAE Mean Absolute Error. 9, 30, 36–38, 40

ML Machine Learning. 1, 3, 4, 8–12, 14, 16, 17, 20, 22, 25, 30, 38, 45–47

MSE Mean Squared Error. 30

RAM Random Access Memory. 14, 31

SGD Stochastic Gradient Descent. 10, 30

B. Appendix - Code Listings

The Dataset acquired in this project and all of the code is publicly available at:

https://github.com/LaurenzBeck/Vision-Control

For the sake of completeness the code used for training is also listed in this appendix:

Training script used for the model from section 2.2.4.7

```
import sys
assert sys.version_info >= (3, 5)

import tensorflow as tf
assert tf.__version__ >= "2.0"
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
    , Conv2D, MaxPooling2D
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
from time import time
import datetime

%matplotlib inline

SEED = 42
SESSION_NAME = datetime.datetime.now().strftime("%Y%m%d_%H%M")
PI = tf.constant(np.pi)
```

```
24
25 # Loading the DataFrames
26 df_train=pd.read_csv("./labels_train.csv",dtype={'id': str, 'label': np.float32})
27 df_val=pd.read_csv("./labels_val.csv",dtype={'id': str, 'label': np.float32})
28 df_test=pd.read_csv("./labels_test.csv",dtype={'id': str, 'label': np.float32})
29 IMG_PATH_TRAIN = "images_train/"
30 IMG_PATH_VAL = "images_val/"
31 IMG_PATH_TEST = "images_test/"
32
33 df_train.head()
34
35 # Visualizing some examples
36 fig = plt.figure(figsize=(10,5))
37 for i in range(1, 5):
38     a = fig.add_subplot(2, 2, i)
39     imgplot = plt.imshow(mpimg.imread(IMG_PATH_TRAIN+df_train.iloc[(i*10000)-1]['id']),
           cmap='gray', vmin=0, vmax=1)
40     a.set_title(df_train.iloc[(i*10000)-1]['label'])
41     plt.tight_layout()
42
43
44 # Data Augmentation Pipeline
45 HEIGHT = 480
46 WIDTH = 640
47 TARGET_HEIGHT = 480
48 TARGET_WIDTH = 640
49 CHANNELS = 1
50 BATCH_SIZE = 16
51 NUM_TRAIN_IMAGES = 80000
52 NUM_VAL_IMAGES = 10000
53 NUM_TEST_IMAGES = 10000
54
55 datagen = ImageDataGenerator(rescale = 1/256,
56                              width_shift_range = 25,
57                              height_shift_range = 25,
58                              rotation_range = 5,
59                              brightness_range = [0.7, 1.3],
60                              zoom_range = 0.25)
61
62 train_generator = datagen.flow_from_dataframe(dataframe=df_train,
63                                                directory=IMG_PATH_TRAIN,
64                                                x_col="id",
65                                                y_col="label",
```

```
                                                seed=SEED,
                                                batch_size=BATCH_SIZE,
                                                shuffle =True,
                                                class_mode="raw",
                                                target_size =(TARGET_HEIGHT,
                                                    TARGET_WIDTH),
                                                color_mode = "grayscale")

val_generator = datagen.flow_from_dataframe(dataframe=df_val,
                                            directory=IMG_PATH_VAL,
                                            x_col="id",
                                            y_col="label",
                                            seed=SEED,
                                            batch_size=BATCH_SIZE,
                                            shuffle =False,
                                            class_mode="raw",
                                            target_size =(TARGET_HEIGHT,
                                                TARGET_WIDTH),
                                            color_mode = "grayscale")

test_datagen = ImageDataGenerator(rescale = 1/256)

test_generator = test_datagen.flow_from_dataframe(dataframe=df_test,
                                                  directory=IMG_PATH_TEST,
                                                  x_col="id",
                                                  y_col="label",
                                                  batch_size=BATCH_SIZE,
                                                  class_mode="raw",
                                                  shuffle =False,
                                                  target_size =(TARGET_HEIGHT,
                                                      TARGET_WIDTH),
                                                  color_mode = "grayscale")

# Visualizing augmented images
fig = plt.figure( figsize =(12,10))
for X_batch, y_batch in train_generator:
    for i in range(1, 9):
        image = ((X_batch[i-1]*256).astype('uint8')) [:,:,0]
        a = fig.add_subplot(4, 2, i)
        imgplot = plt.imshow(image, cmap='gray', vmin = 0, vmax = 256)
        a. set_title ("{}".format(i))
        plt . tight_layout ()
    break
```

```
106
107
108 # Model definition
109 model = keras.models.Sequential([
110     Conv2D(32, 7, activation="elu", padding="same", kernel_initializer ="he_normal",
111                             input_shape=[TARGET_HEIGHT,TARGET_WIDTH,CHANNELS]),
112     Conv2D(32, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
113     BatchNormalization(),
114     AveragePooling2D(pool_size=2),
115     Conv2D(16, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
116     Conv2D(32, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
117     BatchNormalization(),
118     AveragePooling2D(pool_size=2),
119     Conv2D(16, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
120     Conv2D(32, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
121     BatchNormalization(),
122     AveragePooling2D(pool_size=2),
123     Conv2D(32, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
124     Conv2D(64, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
125     BatchNormalization(),
126     AveragePooling2D(pool_size=2),
127     Conv2D(32, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
128     Conv2D(64, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
129     BatchNormalization(),
130     AveragePooling2D(pool_size=2),
131     Conv2D(64, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
132     Conv2D(128, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
133     BatchNormalization(),
134     AveragePooling2D(pool_size=2),
135     Conv2D(64, 1, activation="elu", padding="same", kernel_initializer ="he_normal"),
136     Conv2D(128, 3, activation="elu", padding="same", kernel_initializer ="he_normal"),
137     BatchNormalization(),
138     GlobalAvgPool2D(),
139     Dense(32, activation="elu"),
140     Dropout(0.2),
141     Dense(16, activation="elu"),
142     Dense(1, activation="relu")
143 ])
144
145 model.summary()
146
147 # Custom loss function
148 def squared_angdiff_and_mse(y_true, y_pred):
```

```
149     squared_angdiff = tf.square( tf.atan2(tf.sin(y_pred - y_true), tf.cos(y_pred - y_true)) )
150     mse = tf.square(y_pred - y_true)
151     loss = tf.add(squared_angdiff, mse)
152     return loss
153
154 # Learning rate scheduler
155 def scheduler(epoch):
156   if epoch < 3:
157     return 0.004
158   elif epoch < 6:
159     return 0.002
160   elif epoch < 20:
161     return 0.001
162   elif epoch < 30:
163     return 0.0008
164   else:
165     return 0.0005
166
167 # Keras Callbacks
168 schedule = keras.callbacks.LearningRateScheduler(scheduler)
169
170 FILEPATH_SAVE='checkpoints\{}_model_weights.h5'.format(SESSION_NAME)
171 checkpoint = keras.callbacks.ModelCheckpoint(FILEPATH_SAVE,
172                                              monitor=[squared_angdiff_and_mse],
173                                              verbose=1,
174                                              mode='min')
175
176 FILEPATH_TENSORBOARD_LOG = 'logs\{}'.format(SESSION_NAME)
177 tensorboard = keras.callbacks.TensorBoard(log_dir = FILEPATH_TENSORBOARD_LOG,
178                                           histogram_freq=1,
179                                           write_graph=True,
180                                           write_images=True,
181                                           update_freq=10000,
182                                          )
183
184 # Model Compilation
185 model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01,
186                                               beta_1=0.9,
187                                               beta_2=0.95,
188                                               epsilon=1e-07),
189               loss=squared_angdiff_and_mse,
190               metrics=[squared_angdiff_and_mse, 'mae'])
191
```

```
192 # Training
193 NUM_EPOCHS = 45
194 history = model.fit_generator(generator=train_generator,
195                                validation_data=val_generator,
196                                validation_steps=625,
197                                steps_per_epoch=5000,
198                                epochs=NUM_EPOCHS,
199                                callbacks=[tensorboard, checkpoint, schedule])
200
201 # Model Evaluation
202 model.evaluate_generator(generator=test_generator, steps= 625)
203
204 # Log test results
205 test_generator.reset()
206 pred=model.predict_generator(test_generator, verbose=1)
207 filenames=test_generator.filenames
208 labels = []
209 for filename in filenames:
210     labels.append('.'.join([filename.split('_')[1].split('.')[0], filename.split('_')[1].
          split('.')[1]]))
211 labels_f = [float(x) for x in labels]
212 pred_f = [float(x) for x in pred]
213 diff = []
214 for i in range(len(labels)):
215     diff.append(pred_f[i]-labels_f[i])
216 results=pd.DataFrame({"Filename":filenames,
217                       "Label": labels,
218                       "Predictions":pred.tolist(),
219                       "Difference": diff})
220 results.to_csv(" results_train_ {}.csv".format(SESSION_NAME),index=False)
221 results.head()
222
223 # Save trained model
224 model.save("13_11\model_{}.h5".format(SESSION_NAME))
```