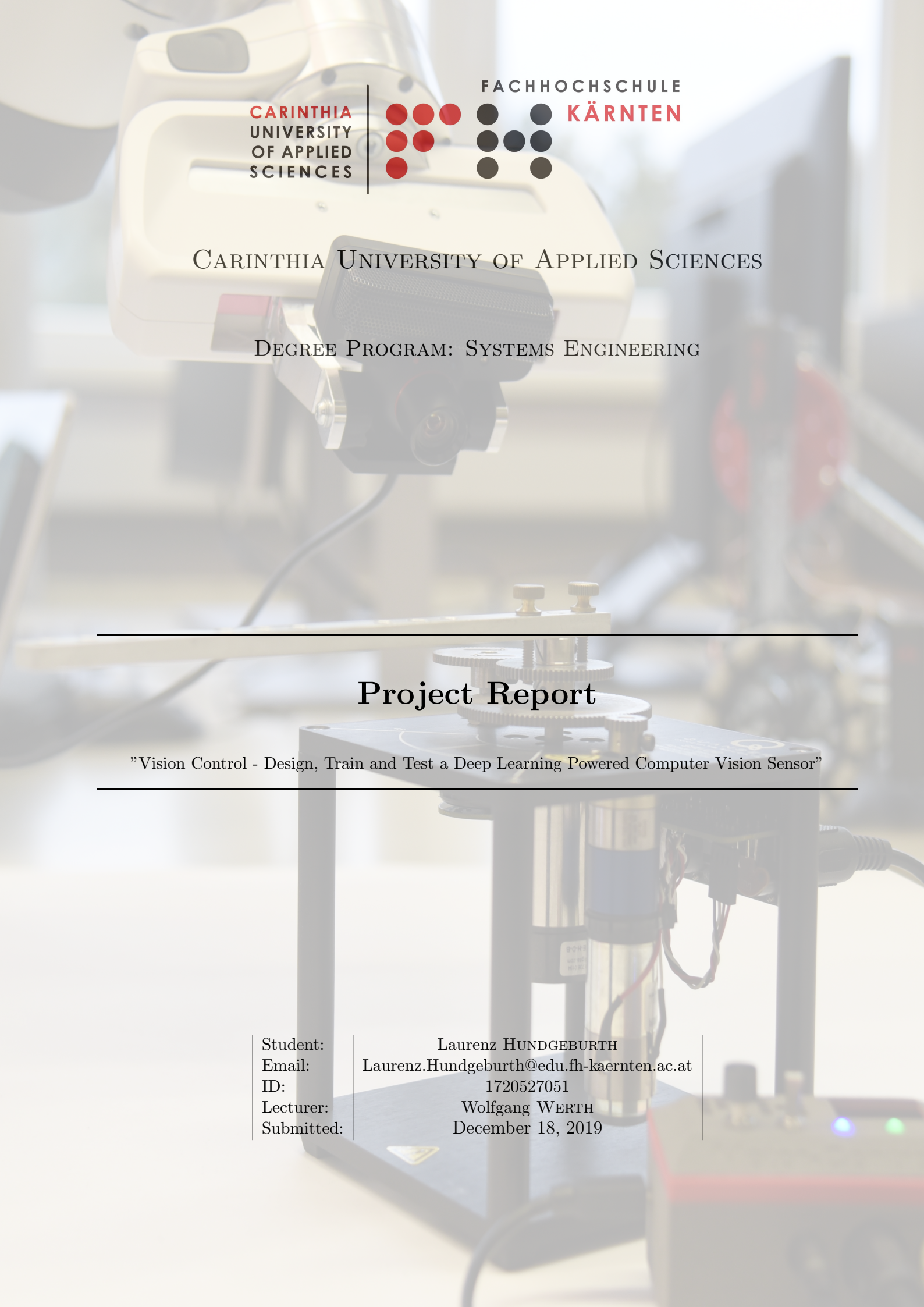


CARINTHIA UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE KÄRNTEN

Carinthia University of Applied Sciences

Degree Program: Systems Engineering

Project Report

"Vision Control - Design, Train and Test a Deep Learning Powered Computer Vision Sensor"

Student:	Laurenz Hundgeburth
Email:	Laurenz.Hundgeburth@edu.fh-kaernten.ac.at
ID:	1720527051
Lecturer:	Wolfgang Werth
Submitted:	December 18, 2019

Contents

List of Figures

1 Introduction

1.1 Motivation and Background

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, yielding more or less sophisticated models able to describe and predict the behaviour of different phenomena. 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 (at least in my opinion more exciting) part of model selection and systems design. Machine learning (ML) offers such an alternative approach as it gives 'machines' (computers programs) the ability to optimize themselves.

The performance and capabilities of ML algorithms were for a long time limited by slow computers and a general rarity of larger data sets. However, this considerably changed during the last decade, as the average working-station is now able to train those algorithms in a practical amount of time. The interest in ML algorithms has also gained a lot of traction, as they have lately proven unbeknownst performance in many difficult tasks including computer vision and speech recognition often even surpassing human performance.

1.2 Project Overview

One of the first challenges was to find an appropriate problem to solve. It had to be something which would be difficult to solve with traditional methods. Moreover, the ease of collecting training data and general availability of the required hardware were also some important factors to consider. Aiming for something simple would set the focus on the ML related problems rather than on poor yielding data acquisition or sophisticated models.

The decision fell for a vision sensor that can be used in control applications. This sensor will measure the rotation angle of a special servo motor available at Carinthian University of Applied Sciences. This project consists of three major steps, including the acquisition of a proper data-set, the training of a Convolutional Neural Network (CNN) and some unit integration tests, where we battle-test our sensor with hardware under different circumstances.

2 Methods

2.1 Data Acquisition

The success of most ML research projects depends on the quality and amount of data used for training [7]. The most sophisticated models are often not enough to compensate for data related deficiencies like a general sparsity of data, an imbalance in the distribution of a data set or other forms of high biased data.

The main goal of the proposed acquisition strategy is, therefore, to mitigate most of the known data-related problems, but also have a good yield due to a high degree of automation.

2.1.1 Hardware and Setup

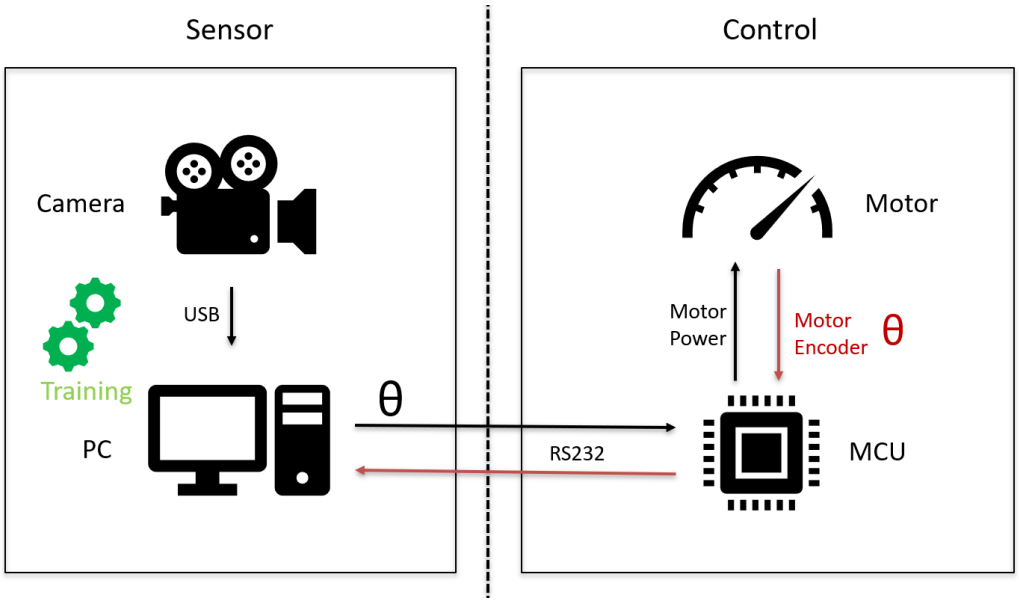


Figure 2.1: Schematic of Data Acquisition Setup

A servo motor block from Quanser [9], 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 microcontroller 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 [8] mounted on a Franka Emika robot[2]. 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 this 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

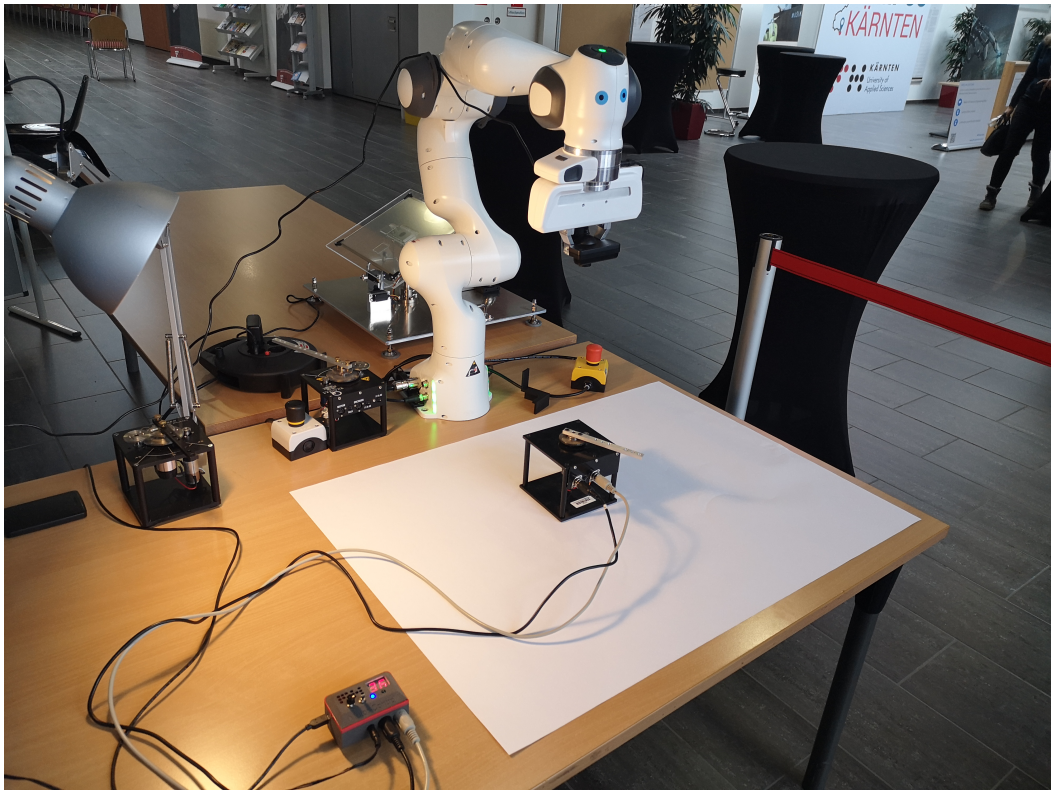


Figure 2.2: Final Setup

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.

With this setup the following sets were acquired:

- **Training set** consisting of 80.000 images
- **Validation set** consisting of 10.000 images with the same camera movement variations as in the training set, but with different backgrounds and lighting settings
- **Test set** consisting of 10.000 images with a different camera movement variation and more dramatic lighting conditions

2.2 Model Selection

With such a big training set it is possible to train a deep neural network which will learn the relationship between the $480 * 640 = 307.200$ individual pixel as inputs (using grayscale), to a single output representing the current motor position.

Choosing a model designed for computer vision tasks like Convolutional Neural Networks can dramatically decrease the number of needed model parameters, while at the same time increasing performance and accuracy. The main idea behind CNNs is to reduce the dimensionality of the data through the means of traditional image processing methods like convolutions and pooling.

Such a model can be easily implemented using Googles ML library TensorFlow [6]. TensorFlow provides a computational back-end which is optimized for ML workloads. The complexity of using TensorFlows low-level API can be reduced by using Tensorflows official high-level API called Keras. The model definition used in this project is depicted in the listing below. The listing is written in Python 3.7 and uses Keras sequential model definition.

```
from tensorflow.keras.layers import Dense, GlobalAvgPooling2D, Dropout, BatchNormalization,
    Conv2D, AveragePooling2D

model = keras.models.Sequential([
    Conv2D(32, 7, activation="elu", padding="same", kernel_initializer="he_normal", input_shape
        =[480, 640, 1]),
    Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(16, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(16, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(32, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(32, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(64, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(32, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(64, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(64, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(128, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    AveragePooling2D(pool_size=2),
    Conv2D(64, 1, activation="elu", padding="same", kernel_initializer="he_normal"),
    Conv2D(128, 3, activation="elu", padding="same", kernel_initializer="he_normal"),
    BatchNormalization(),
    GlobalAvgPool2D(),
    Dense(32, activation="elu"),
    Dropout(0.2),
    Dense(16, activation="elu"),
    Dense(1, activation="relu")
])
```

Model Definition using Keras Sequential Api

2.3 Training

Once a model is selected and a data set is available, the training process has to be set up correctly. This process includes the following parts:

- **data preprocessing**
- **choosing a loss function**
- **choosing an optimizer**

Poor decisions made in one of those parts can have consequences ranging from slowing down training to yielding completely unusable results. In the following subsections, the decisions made for this project

will be explained in more detail.

2.3.1 Data Preprocessing

The first step in the preprocessing pipeline is to scale each pixel value from a range of 0 - 255 to a range of 0 - 1. This serves the purpose of bringing the input values nearer to 0, where the "ELU" activation function [1] used in this project has its non-linearity. That way the model will converge faster and the wights will normally remain small, thus avoiding the "exploding gradients" problem [4].

The next part of the preprocessing pipeline is a variety of image augmentations like height/width shifting, zooming, rotating, brightness shifting and a colour transformation to greyscale. All these augmentations synthetically increase the effective amount of training data and help the model to generalise better.

2.3.2 Custom Loss

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). [11]

This would work to a certain extent but is not optimal at the extreme values. Consider the following problem: 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 we need, a loss function, which also considers the periodic nature of our rotational system. For the given reasons a loss function consisting of both, the MSE and the squared angular difference was chosen.

$$loss = \frac{1}{n} * (\arctan(\frac{\sin(y - \hat{y})}{\cos(y - \hat{y})})^2 + (y - \hat{y})^2) \tag{2.1}$$

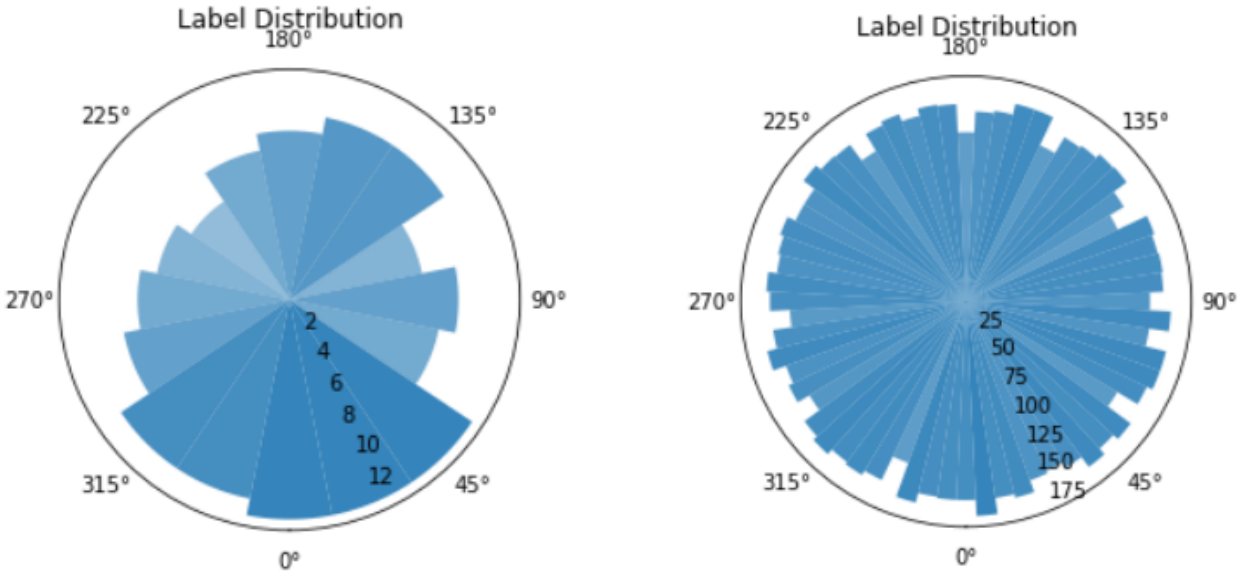
2.3.3 Optimization

There are a variety of different optimization algorithms leading to more or less the same results. Each of those algorithms specialises for different qualities, like the speed of convergence, the quality of convergence (the difference between the absolute minimum and the proposed solution), or the robustness against a tendency to get stuck in local minima instead of global ones.

Using the popular Adaptive Momentum (ADAM) [5] optimizer in combination with a technique called learning rate scheduling [3] utilizes the benefits of different optimizers and often leads to fast convergence with a good quality of the solution. A side effect of using learning rate Scheduling is the

decreased importance of learning rate tuning as there is no need for a decision to optimize this parameter for speed (high learning rate) or quality of convergence (low learning rate). Starting with a high learning rate and a stepwise decreasing it will fulfil both criteria.

3 Results



(a) biased distribution due to unfavourable control

(b) even distribution

Figure 3.1: Label Distribution

The time invested in the careful planning of the data acquisition process and the degree of automation achieved by the used strategy paid off, as we not only obtained a large and diverse data set with an even distribution of training labels, but we did so in a relatively short amount of time (approximately 4 labelled pictures per second) compared to other acquisition methods [10].

To pick one example: instead of powering the motor with a constant voltage, thus letting it rotate freely, we implemented a positional control which allowed us to determine the motor position precisely at each measurement. Doing so allowed us to control the distribution of labels as seen in the polar histogram in Fig. 3.1b. Fig. 3.1a shows the label distribution in an earlier phase of the project, where we had the motor laying on its side. The effect of the gravity led to an uneven distribution of labels, which would introduce some considerable amount of bias into a model trained on such a data set, as some labels were simply more likely than others. Using a closed-loop positional control reduced bias problems during training and saved us a lot of time trying to find solutions to fix them afterwards.

Let us investigate the training results of a training run over 45 epochs on 80 thousand training images which ran for about 2 1/2 days. Table 3.1 shows the model evaluation on the 3 different data sets and Fig. 3.2 depicts the training curve of the MAE on both the training set (orange) and the validation set (blue).

The fast progress slowed down at around epoch 8 and 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

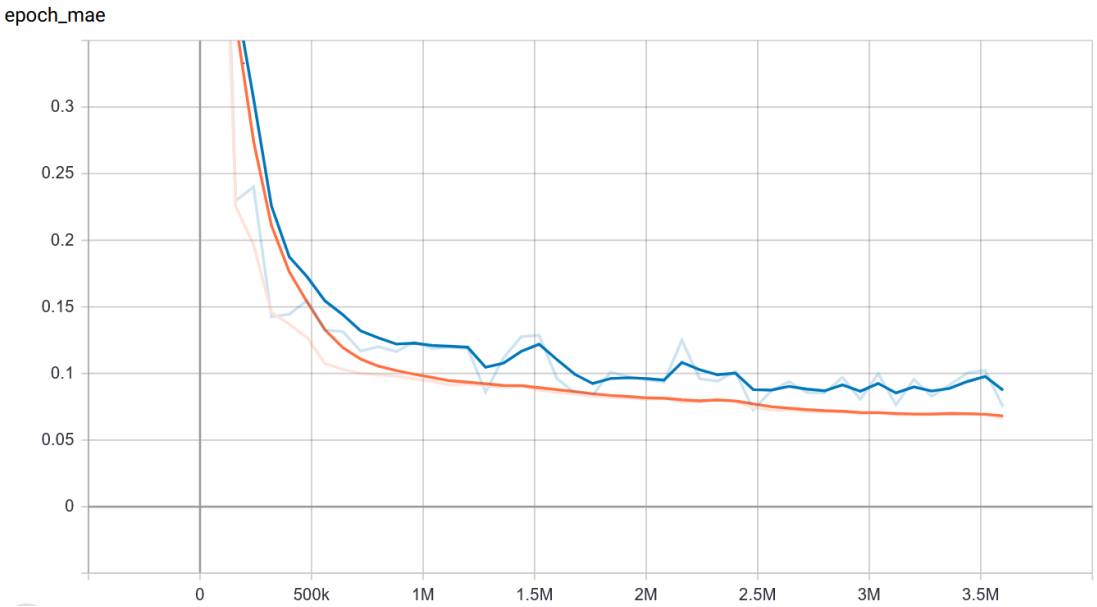


Figure 3.2: Training Curve: MAE

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.

One last interesting result from the project to mention here in this small report is the unexpected minor effect of the chosen model size on the performance, once one has a decent data set to train on. Training runs with models of the same architecture, but with different amounts of filters and neurons led to minor deviations from the training results showed earlier. The variations yielded two models having 15 thousand and 450 thousand parameters respectively which corresponds to a model twice the size of the original one on being 15 times smaller (compared to the original model with its 220 thousand parameters).

Table 3.1: Model Evaluation

metrics	train set	validation set	test set
custom loss (2.1)	0.0554	0.1055	0.0625
mean absolute error	0.0666	0.0747	0.0638

4 Conclusion

There are still a few things to improve, like increasing the difference between the training and validation set. The lighting conditions could also be controlled by the central acquisition script.

It would also be very interesting to design a positional motor control using our model predictions as the only input for control. One would then have to pay special attention to the sensors statistical behaviour. But such an analysis is beyond the scope of this work.

Considering the low expectations we had because of our former lack of ML experience, the trained vision sensor with its acceptable precision and generalization capabilities turned out to be a huge success. A big contribution to that success was our ability to automate the data acquisition, for which our application-focused engineering education can be credited for. This strength of our university should be something to build upon in future projects to leverage more of Machine Learning strengths.

5 Acknowlegment

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.

I am very grateful for Andrew Ng's amazing online courses on deep learning [7] and Aurélien Géron's fantastic book "Hands-on Machine Learning" [3], 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.

6 Bibliography

[1] Clevert, D.-A., Unterthiner, T., and Hochreiter, S., "Fast and accurate deep network learning by exponential linear units (elus)", Jan. 2016.

[2] *Franka emika robot*. [Online]. Available: `https://www.franka.de/`.

[3] 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.

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

[5] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization", *CoRR*, vol. abs/1412.6980, 2014.

[6] Martın 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/`.

[7] Ng, A., *Machine learning — coursera*, 2019. [Online]. Available: `https://www.coursera.org/learn/machine-learning`.

[8] *Playstation eye camera*. [Online]. Available: `https://en.wikipedia.org/wiki/PlayStation_Eye`.

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

[10] Roh, Y., Heo, G., and Whang, S., *A survey on data collection for machine learning: A big data - ai integration perspective*, Nov. 2018.

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