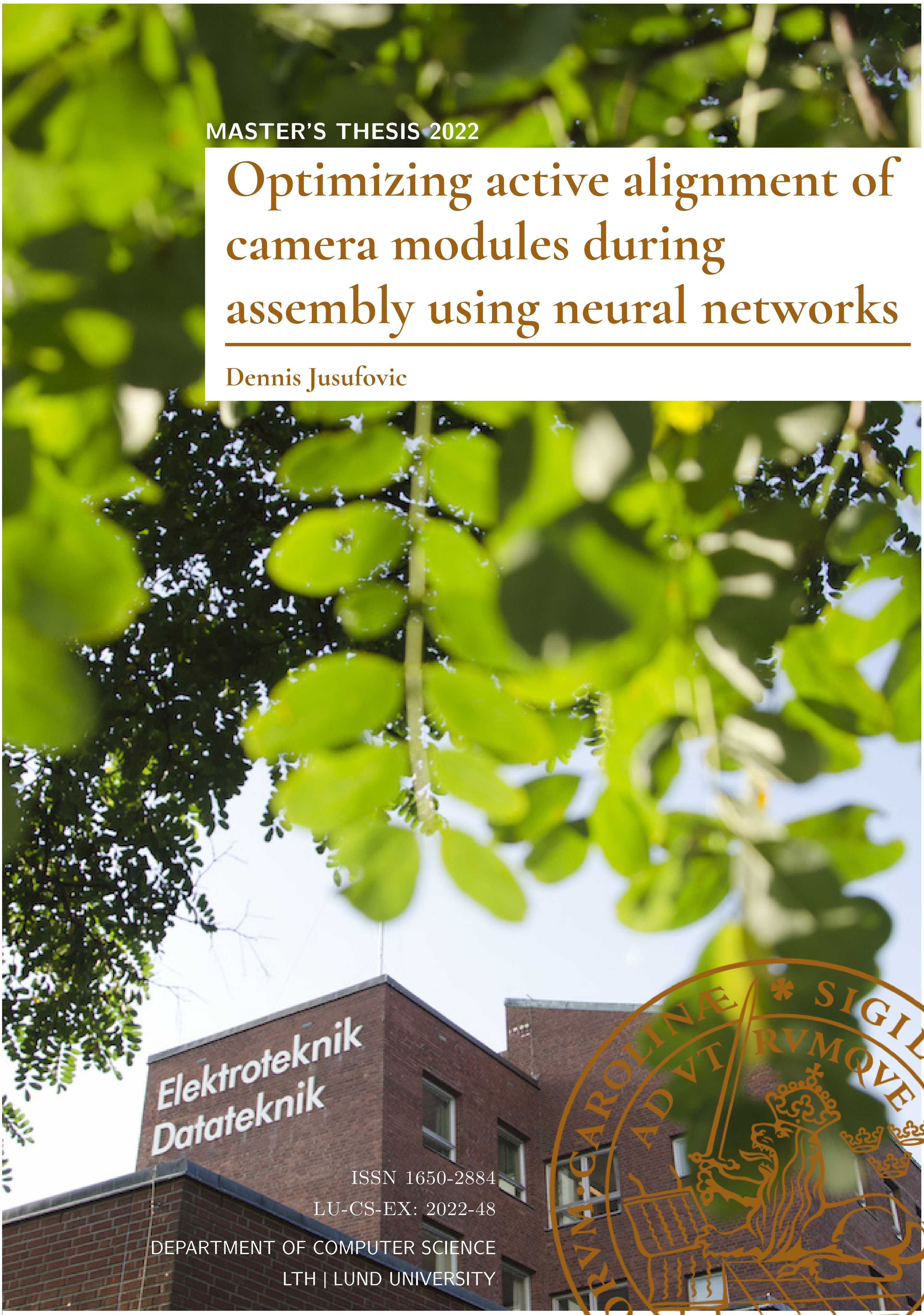

MASTER'S THESIS 2022

Optimizing active alignment of camera modules during assembly using neural networks

Dennis Jusufovic

ISSN 1650-2884
LU-CS-EX: 2022-48

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-48

Optimizing active alignment of camera modules during assembly using neural networks

Optimering av "active alignment" av kameramoduler i produktion med neurala nätverk

Dennis Jusufovic

Optimizing active alignment of camera modules during assembly using neural networks

Dennis Jusufovic
de5033ju-s@student.lu.se

June 20, 2022

Master's thesis work carried out at Axis Communications.

Supervisors: Dr. Dennis Medved, dennis.medved@med.lu.se
Martin Hellaeus, martin.hellaeus@axis.com
Dr. Stefan Plogmaker, stefan.plogmaker@axis.com

Examiner: Prof. Jacek Malec, jacek.malec@cs.lth.se

Abstract

When a camera module is assembled a crucial step is the joining of the lens and sensor. If the sensor is not at the correct distance or angle in relation to the lens the module will perform badly e.g., will get bad focus. This is a complex problem since the optimal distance or angle can vary from lens to lens. At Axis the problem is solved with active alignment. Each lens and sensor pair is analyzed until the most optimal distance and angle are found, resulting in saved costs and increased production yield.

In this study we try to optimize Axis' active alignment process using artificial neural networks. We explore three different model candidates and compare their performance. Our experiments show that image analysis using neural networks would be a good replacement for the current approach, but that more work should be done on making the model more generalizable.

Keywords: MSc, active alignment, Axis, passive auto focus, machine learning, deep learning

Acknowledgements

I would like to thank my supervisor at Lund University, Dennis Medved for giving me feedback and guidance. A special thanks to Martin Helleaus at Axis who helped me collect the data sets and Stefan Plogmaker who together gave me the opportunity to take part in this interesting study.

Contents

Chapter 1
Introduction

Machine learning, there are no two words in computer science that spark as much interest to the outside world. It brings promises of what could only have been called science fiction a mere 10 years ago. Take image captioning, which would not have nearly the same success today without this field. The same goes for music generation. Problems like these seem like there where made to be solved by machine learning, but it is not always that simple. It is easy to get caught up in the hype. If you only have a hammer every problem will start to look like a nail. What about problems we have already solved with traditional solutions? Could perhaps machine learning prove to be a better solution? Or are we just desperately trying to bang a screw with our new shiny hammer?

In this study we are faced with the latter scenario. Axis or indeed almost every camera manufacturer uses a method called active alignment during assembly. We will determine if we can optimize the active alignment process using different artificial neural networks and compare the results.

1.1 Active Alignment

Every assembly part is not perfect. There will always be tolerances, that is just the reality of manufacturing. This is especially true with lenses and sensors. One way of achieving high performance despite of this is to have adjustable lenses. That comes with its own set of problems. It increases complexity, cost and becomes more and more impractical with the increased demand for small and compact camera modules, for example phone cameras. Another option is to buy parts with very small tolerances that we know would be good enough. The disadvantage is that high precision made parts are very expensive.

Active alignment solves this by taking the small variations into account when merging the lens and sensor. Finding the optimal tilt, translation, distance, and generating good performance.

For our purposes we can simplify the camera module as consisting of two parts the lens

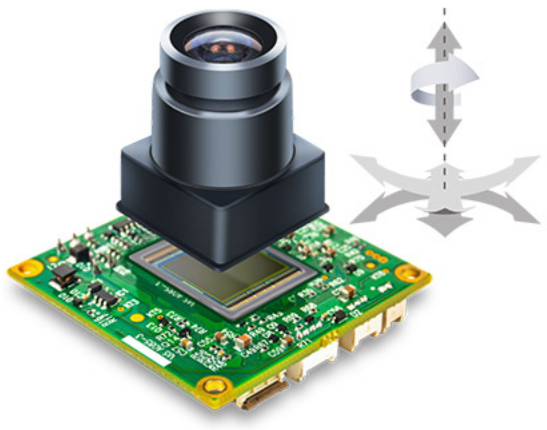

Figure 1.1: A lens module and sensor chip [17].

module and the sensor chip (Figure 1.1). During the alignment both the sensor and lens are connected to a test machine. This means that we can receive images in real time from the sensor during the entire process. Collimators are placed above the lens which project an image to the sensor in the shape of a cross (Figure 1.2). This cross will be the target.

The lens will be moved towards the sensor along the z-axis. The goal is to find the distance where we have the best focus of the target. Currently the focus peak is found by moving the sensor one step at a time evaluating each picture until a peak has been detected. This is not the entire process just one step called coarse alignment, but it is enough for the scope of this thesis.

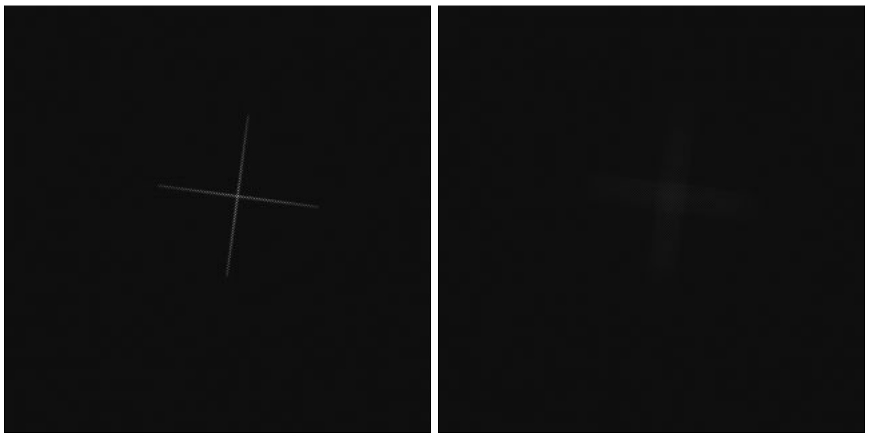

Figure 1.2: The image taken by the sensor during active alignment. The left picture shows when the target is in focus and the right shows when it is not.

1.2 Problem formulation

As we mentioned, we are mostly concerned about the coarse alignment step of the active alignment. The method of searching for a focus peak step by step from start to finish is very simple but slow. It requires around 20-30 images to be taken. This creates a bottleneck because images take a relatively long time to get loaded into the computer. There is definitely room for improvement by reducing the amount of images that needs to be taken. Our problem formulation is as follows:

- Is there a way to use machine learning or more specifically artificial neural networks to reduce the time searching for the focus peak?
- How well do different types of neural network fare in this task?

1.3 Previous Work

There has been work done on applying machine learning to the entire active alignment process [6]. The study was only aligning FAC-lenses not lens to camera modules like we are trying to do. Even so they concluded that the machinelearning-based approach is a promising alternative to a model-based one.

Unfortunately there is not much more research to be found on this topic, but as said before we are not looking at a the whole active alignment process. Since we have limited ourselves to coarse alignment we can generalize the problem further. What we are really trying to do is to determine if a image is in focus and if it is not, at what distance should we find it. This is what auto focus does, something we all have come to expect from most cameras we use. This very much broadens the research available to us since this topic has been studied far more.

1.3.1 Active vs Passive Auto-Focus

Auto-focus can be divided in two different categories, active and passive [2]. Active auto-focus uses information from additional sources than the image sensor. This could be measured using infrared rays or an ultra sonic component.

Most modern DSLR and smartphone cameras use passive auto-focus [1], where proper focus is determined only by the image sensor. Modern Samsung phones use something called on-sensor Dual Pixel auto-focus [14], trying to match the phase of two different light rays from the same point [1]. While this method is not possible for us to use since it requires a special sensor our approach of analyzing the images with a neural network still falls into this category.

1.3.2 Passive Auto-Focus using Deep Learning in Microscopy

The work done in microscope passive focusing most closely resembles our use case. In both cases a stationary camera is pointed at a predictable target with limited movement. In a study [13] the authors managed to create a neural network based on a pretrained model called MobileNetV2 [15] that could accurately predict the optimal focus with a difference image of two images close to each other as input (Figure 1.3). They used images taken across a set interval over the focus peak on around 100,000 samples.

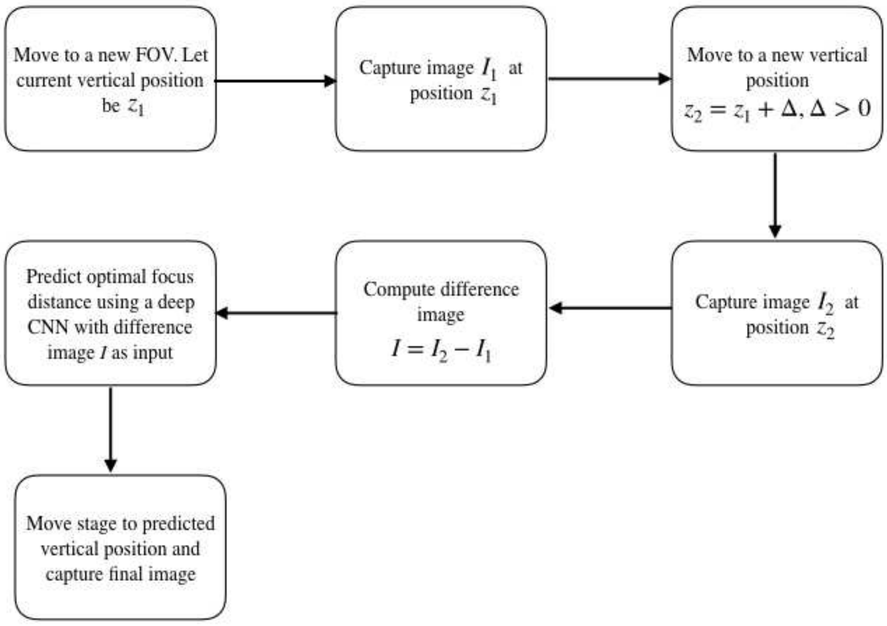
Move to a new FOV. Let current vertical position be z_1
Capture image I_1 at position z_1
Move to a new vertical position $z_2 = z_1 + \Delta, \Delta > 0$
Capture image I_2 at position z_2
Compute difference image $I = I_2 - I_1$
Predict optimal focus distance using a deep CNN with difference image I as input
Move stage to predicted vertical position and capture final image

Figure 1.3: A illustration of the proposed method [13].

Another study [12] found machine learning suitable for this task. Roughly put, they had a more advanced model than the previous group where they applied a Fourier transformation to the image before they put it through a neural network 1.4. Illumination from different sources was key since the features was still sharp even though they were out of focus.

Intuitively we seem to have a more simple problem. Our target will always be the same, a white cross. The models described above where able to handle focusing using images with different features and also taking color into account.

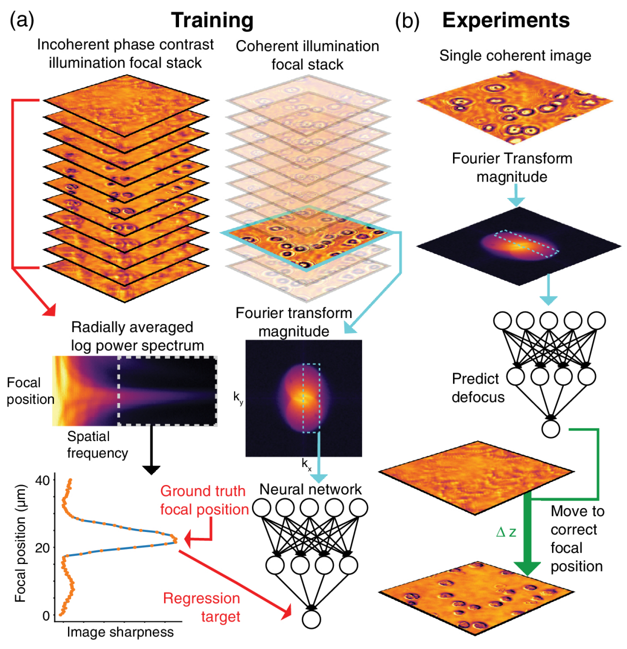
(a)
Training
Incoherent phase contrast illumination focal stack
Coherent illumination focal stack
Radially averaged log power spectrum
Focal position
Spatial frequency
Fourier transform magnitude
k_y
k_x
Focal position (μm)
40
20
0
Image sharpness
Ground truth focal position
Regression target
Neural network
(b)
Experiments
Single coherent image
Fourier Transform magnitude
Predict defocus
Δ z
Move to correct focal position

Figure 1.4: A illustration of the proposed method [13].

Chapter 2

Background Information

In this chapter we will go though all the necessary information required to get a good understanding of the study. The target audience is presumed to have a general knowledge of computer science but not necessarily any experience with artificial neural networks.

2.1 ANN - Artificial Neural Networks

We find it necessary to clear up some common misunderstandings about the terms machine learning, deep learning and artificial neural networks. Artificial intelligence is the broadest definition and encompasses machine learning. Machine learning is a super set of deep learning (Figure 2.1) as well as neural networks. Though in the last decade ANN have become more and more synonyms with deep learning since most networks today are deep [5].

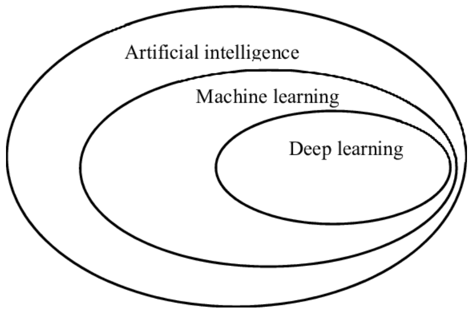

Figure 2.1: A venn-diagram depicting the relation between the fields [10].

2.1.1 The ANN Node

"Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level." [9]

Let us start with the most simple building block of a neural network, a node (Figure 2.2). The idea comes from the neuron in our brains [11]. It takes a variable number of inputs where each input is weighted by a unique factor (ω). The weighted inputs are summed and a bias term (b) is added. This is passed to a activation function (φ) which gives the output (Equation 2.1 - 2.2).

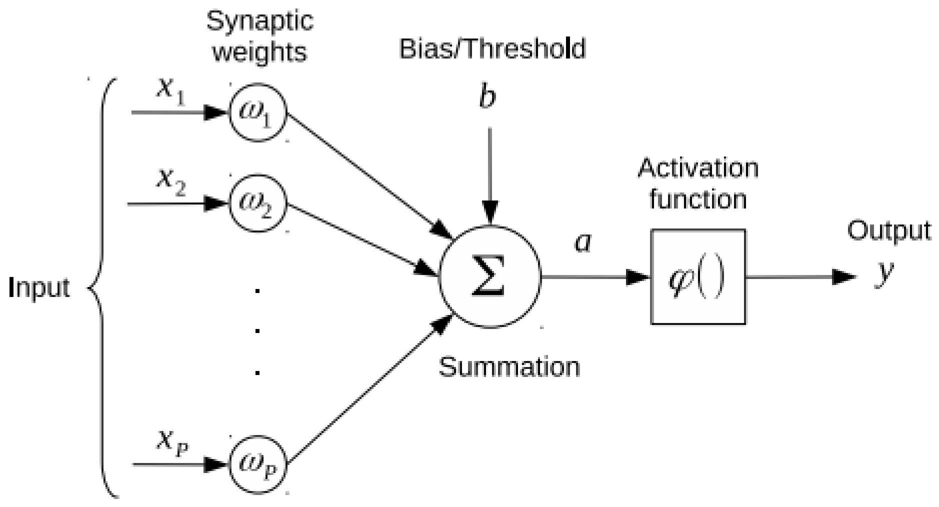

Figure 2.2: A illustration of a single ANN node [11].

$$a = \sum_{k=1}^{K} \omega_k x_k + b \tag{2.1}$$

$$y = \varphi(a) \tag{2.2}$$

Artificial neural network nodes are the basic building block that all our networks will consist of. There are many ways to structure them, but what we will be using in this study can be categorized as feed forward networks.

2.1.2 Feed Forward Networks

The feed forward network is a very common type of network and has found a lot of success [16] in the recent decades. The nodes are structured in groups called layers (Figure 2.3). There is a input layer in the beginning, a variable number of hidden layers in the middle and a output layer in the end. No nodes in the same layer are connected to each other, and the data may only flow from the input layer towards the output layer. That is where the name feed forward comes from. The parameters in the hidden layers are just that, hidden. We have no way of directly interfacing with those layers [11], which has the side effect that it is very difficult to reason out how a network makes decisions.

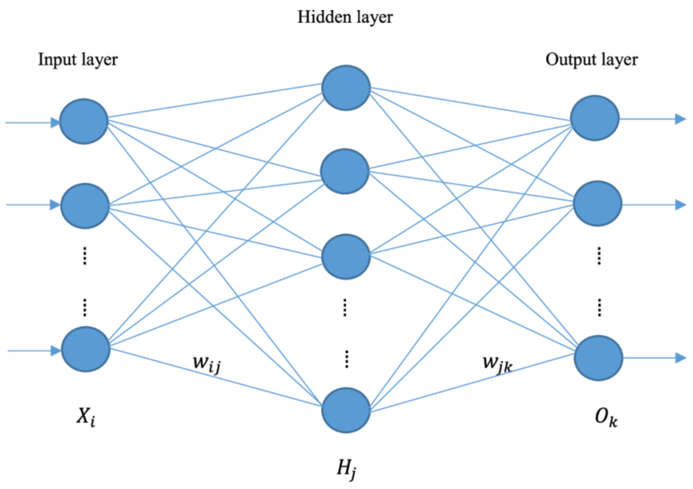

Figure 2.3: Visual graph of a feed forward neural network [20].

2.1.3 Training

Training a model means trying to find the weights ω and biases b that solves the problem best. How do we define what is best? A loss function will need to be defined. Often with every sample from a data set a expected value is included. The loss functions that we will use in this study is the mean absolute error and mean squared error (Equation 2.3). The N is the number of samples y_n is the predictions and d_n is the targets.

$$E(\omega) = \frac{1}{2N} \sum_{n=1}^{N} (y_n - d_n)^2 \tag{2.3}$$

Now we can use the loss function to calculate new weights and biases. The most common method is called gradient decent. When using mean squared error the formula for updating a weight ω_k is as follows:

$$\omega_k = \omega_k + \eta \sum_n \delta_n x_{nk}$$

Where:

$$\delta_n = \frac{1}{N}(d_n - y_n)$$

And η is a factor called the learning rate. It can have a big impact on the performance of a network.

2.1.4 Activation functions

Studies have shown that the most important parameters can be found in the output layer [16]. Meaning that it is very important to choose the correct activation function for those nodes. There are many different different activation functions that serves different purposes. The only ones we will use in this study are the ReLU and linear activation functions.

ReLU

ReLU is a simple function that passes through all arguments except negative ones which give an output of zero (Figure 2.4). It is used as activation functions in the hidden layers [11] because it is simple and fast to compute. Since the activation layer of the hidden layers are not as important as that of the output layer's [16] this compromise in complexity does not have a significant impact on the result.

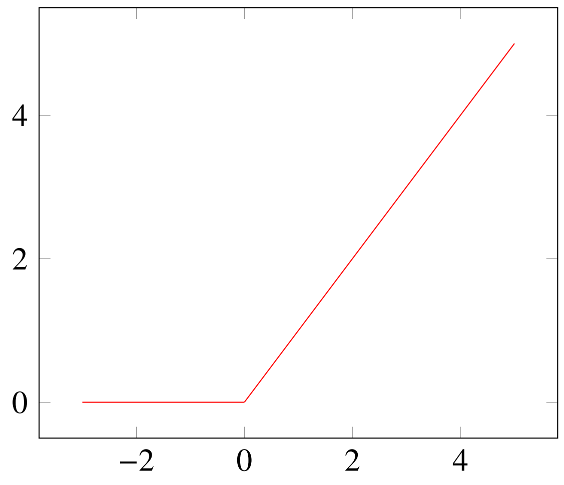

Figure 2.4: A graph of the ReLU activation function.

Linear

The linear activation function is the most simple one. It is the same as no activation, the input given is passed out (Equation 2.4). This function is suitable for output layers when using regression models [13].

$$\varphi(a) = a \tag{2.4}$$

2.1.5 CNN - Convolutional Neural Networks

Convolutional neural networks are the go to network for image recognition and classification. What make them so effective is that they take the space into account. With this method the neighbors of a node are part of the evaluation [11]. The first use of a CNN occurred in 1998 where a group of researchers where able to classify images of hand-draw digits [8].

If we were to use normal dense layers where one pixel is one node then with a standard 2MP picture we would have 2 million nodes. Add another layer where every node is connected to our previous 2 million and we will quickly have too many trainable parameters to handle. What CNNs helps us with is that it reduces the amount of information we have to deal with. It will apply different randomly generated filters to extract features from the image, like edges or textures. In Figure 2.5 we see examples of different images filters that are used in edge detection. Part of the training of this network is to figure out which filters reveal the most information for the network.

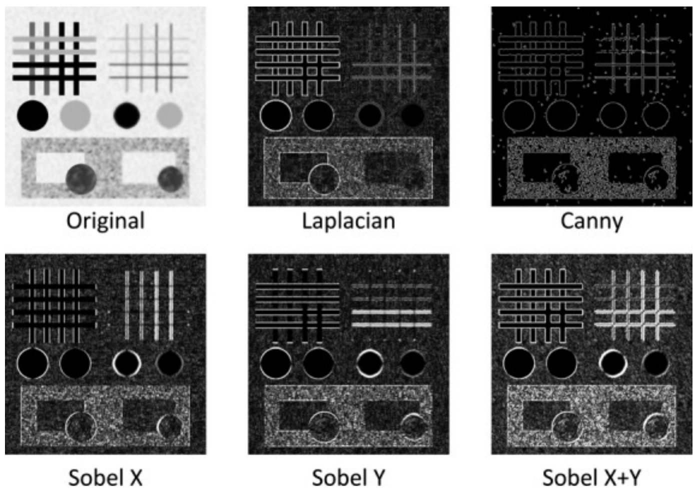

Figure 2.5: Examples of edge detection filters 2.5.

Filtering means applying a convolution with a randomly generated kernel on all pixels of an image. It is easier to describe visually, take Figure 2.6 for an example. Here we have a kernel of size 3 by 3, see Equations 2.5 - 2.6 for how the highlighted pixel gets calculated. The images get element-wise multiplied by the kernel matrix and afterwards summed up. The result will be passed through a activation function, which in this case is ReLu meaning all negative values get set to zero.

$$X = \begin{bmatrix} -1\times 2 & 1\times 1 & 0\times 0 \\ 0\times 1 & 1\times -1 & -1\times 0 \\ 0\times 0 & 0\times 2 & 1\times 1 \end{bmatrix} = \begin{bmatrix} -2 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\sum_i x_i = -1 \tag{2.6}$$

The image is then downsampled using a pooling layer. In Figure 2.6 this is done with a max pooling layer, meaning that a square of nodes gets reduced to the maximum value found in that square [19]. Average pooling is also a quite common method [11].

This process can be repeated with deep learning especially, creating combinations of filters on top of each other and continuing the downsampling. Our trainable parameters are the weights going to a output layer and all the different filter kernels.

2.2 Supervised and Unsupervised Learning

Depending on the data that is available there are two different ways of training a network and that is with supervised or unsupervised learning. In supervised learning, which will be used in this study, we have access to the expected result for a given sample. A common example would be image classification. Imagine a network that needs to differentiate between images of dogs and a cats. In supervised learning the data set would include the correct classification

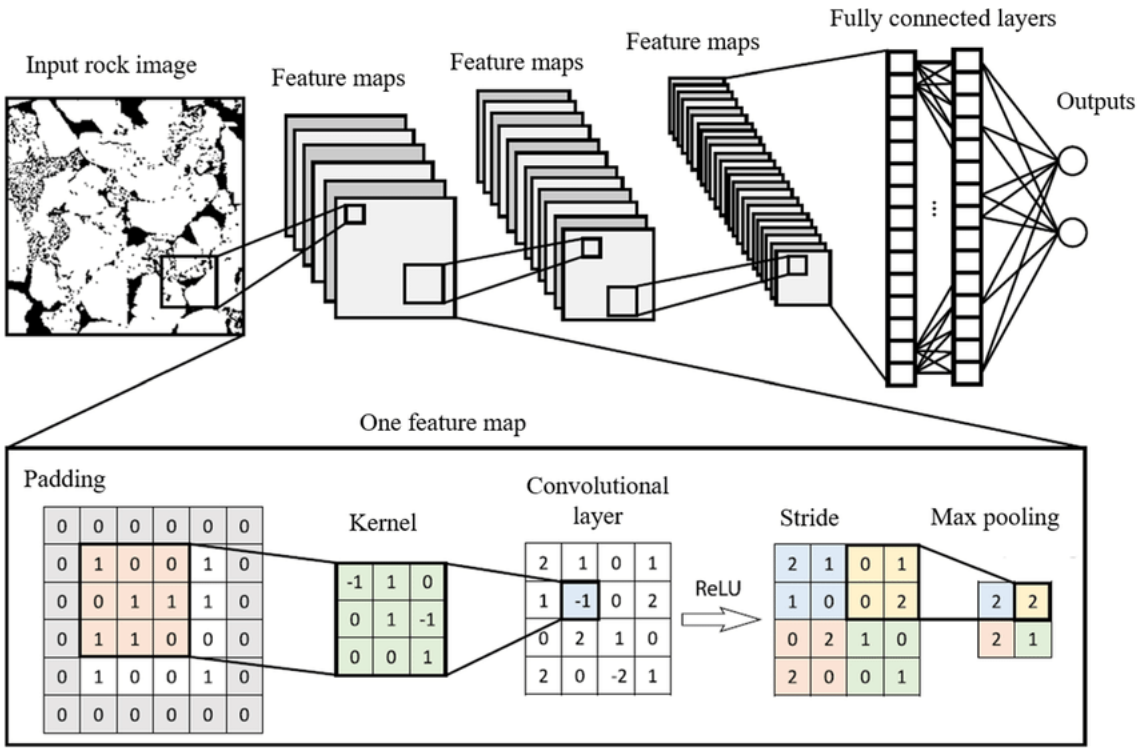

Figure 2.6: A CNN broken down into it filtering, activation and pooling steps [19].

of the images. This is called labeled data and is usually preferred because it results in better performance [11].

If label data is not available we have to turn to unsupervised learning. The network will now have to do more heavy lifting during training. There is simply less information to use. Instead of already having set classes that the model can decide between with unlabeled data the model will have to define its own groups [3]. This is called clustering. Because we do not know during training whether a prediction was correct or not we can only see if these groupings contains similar samples.

Knowing this it is quite easy to understand why supervised learning yields better performance. If a human needed to learn to classify something it would be easier to get instant feedback whether a prediction was true or not instead of having to create their own groups and deciding if they reflect the differences between the classes the best way.

2.3 Transfer Learning

Up until now we have described the training of neural networks under the assumption that the training data set will come from the same source as the target data set. E.g., if we train an image classification network with cats and dogs we expect it to be used on images with cats and dogs. What if that is not entirely necessary? How well would such a network do if it got images of other animals? There should still be a lot of similarities between those tasks.

This is what is called transfer learning, when the training data is taken from a different domain then the target data [18]. It can be useful when there is not enough data to train a network from scratch. A already pretrained network which solves a similar problem could be used as a starting point. It is possible to freeze specific layers, excluding them from training. The output layer can often be removed entirely and replaced with something else, changing the number of classes or even changing a classification problem to a regression one.

Chapter 3
Approach

Our time spent on this study can be divided in two parts, the gathering of data and the creation of the models.

3.1 Data

There is a huge emphasis on getting enough good-quality data. It means that it is not biased, it covers all the edge cases, there are not many outliers and there is enough samples to be able to train the model. Not having enough data can make or break the entire project.

3.1.1 The Testing Machine

The machine that performs the active alignment will be referred to as the testing machine in this report, and it is what we will use to collect our data. It has the capability of determining the optimal tilt, distance and translation and with that in mind glue them together.

The sensor and lens are fastened on two different holders (Figure 3.1) which can be moved and tilted independently of each other. As explained before the sensor and lens are also connected to the testing machine's computer allowing images to be captured from the sensor.

3.1.2 Data Collection

There was no data available that we could use at Axis before the project started, everything had to be collected by us as part of the study. There were two testing machines available for us to use.

We limited the data collection to just one single camera model since this would give us the most consistent data. The chosen camera model is made by the tens of thousands. If we would get the possibility to collect images from production it would provide us with a

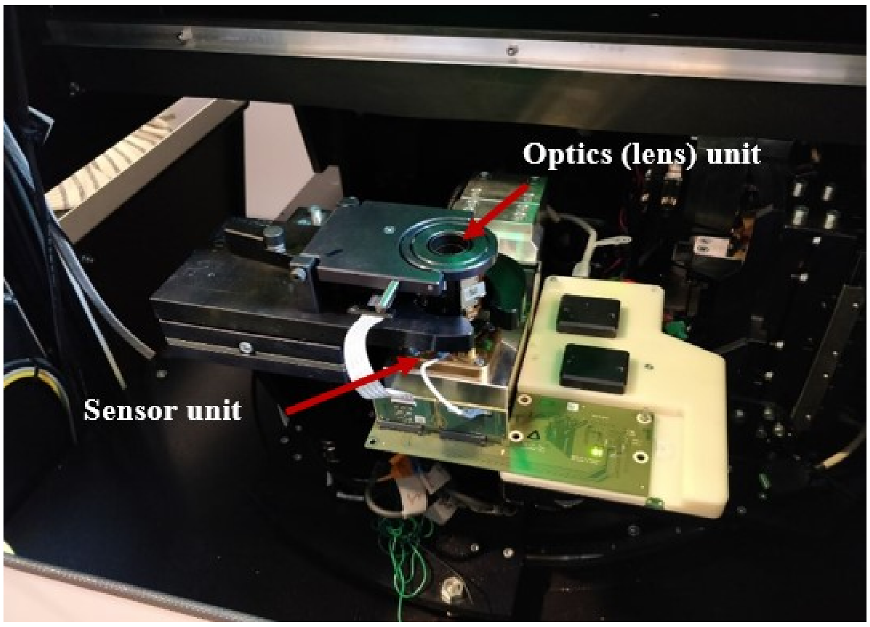

Figure 3.1: A optical module and a sensor on a testing machine, mounted and ready to begin the active alignment process.

lot of data. One other reason is that it uses only the center collimator for coarse alignment, meaning that there is only one cross to focus on 1.2. This gives us a easy target to use in our models compared to four crosses in each corner that other models use.

We got access to six camera modules, meaning six optical modules and six sensor chips. These have already been through the active alignment process and are therefore already glued together, so we had to separated them. This made it possible for us to re-run the active alignment multiple times without gluing them together and also be able to mix and match the lens and chips. This gives us 36 unique combinations of camera modules we can test.

The lens module and sensor chip are placed in there respective holders on the testing machine (Figure 3.1). We had a custom test sequence for the coarse alignment, it went as follows.

1. The distance between the sensor and lens was reset to the maximum distance. We will call this the starting position.

2. Other starting parameters where randomized, particularly the tilt and translation in both the X and Y axes.

3. The lens module was moved one micrometer in the Z axis towards the sensor.

4. A picture was taken with the sensor and is saved on a server. A focus score is calculated based on the image. It gives a value of how well the target is within focus. Because we begin in the starting position we always start moving towards the focus point, meaning that the focus score will increase for each step we take.

5. This process of moving the module and calculating a focus score is repeated until the focus score starts to drop significantly. When that happens we know that we have

passed a peak and can choose the position with the highest focus score as our correct focus distance.

This process will be refereed to as a run. A run gives us a little more then 600 images. For every lens and sensor combination we run eight runs. With 36 combinations, this leaves us with 288 runs or approximately 172800 images.

As we said before we have two testing machines in the lab, and each machine can find the same peak at different distances from its starting position depending on multiple factors e.g. calibration. That is why we will run the 288 runs on two different machines each giving us 576 runs or around 345600 images.

Outliers

It is important for the performance of the trained network to remove as many outliers in the data set as possible. Fortunately for us this is already solved by the fact that these six camera units we are using have already passed quality control. If something is wrong with the focus scores or the cameras are not mounted properly to the testing machine an error will be thrown and the test sequence will fail. We will therefore not have to actively eliminate the outliers.

3.1.3 PGM Image Format

The PGM (Portable Gray Map) image format is a extremely simple grayscale format that is very easy to implement [7]. For reasons we have not been able to figure out the built in Keras load images operation results in a blank image. Therefore we had to implement our own pgm reader to be able to import our test data into the python models.

3.1.4 Datasets

All data used was divided into three types of datasets: training, validation and test set. The training dataset consists of 60 % of all the data and was used as input during training. The validation set which is made up by 20 % of the data was used to test the model between each epoch. After the training the model with the lowest validation loss was chosen as our best model. This is the model that we will evaluate using our testing set.

3.2 Modeling

In this study we experimented with three different types of models. They use different input but all of them output the predicted focus distance either in absolute measurements or relative to the sensor position. All our models will be built and trained using the Keras framework.

The machine used to train these models had a Ryzen 5 3600 processor and a Nvidia 3070 GTX graphics card.

3.2.1 Hyperparameter Tuning

Loosely put hyperparameters are all the parameters that are not trained [4]. In our case it is all the activation functions, layer dimensions, optimizers, learning rates, filters, number of epochs and more. We will need to come up with the best values based on experimentation. There are some formalized methods that are common for example, grid and random search.

Since we do not have a lot of hyperparameters we will just use a loose systemic approach to hyperparameter tuning, experimenting with different values and observing any change in performance. Creating neural models can often be viewed more of an art then a science, it was somewhat true in this case.

3.2.2 Tri-Image model

With this model we wanted to test how well we could predict the focus point using three images taken in fixed positions. There were three image analysis alternatives. First we tried convolutional networks, flattened dense network and some pretrained networks included in keras e.g. efficientnet and mobilenet. We also experimented with using the focus score for each of the images to see if that information would have any impact on the performance.

Because we are using three fixed images we can only use three samples per run. This limits us to a maximum of 288 samples to train with.

During training we used adam as the optimizer and mean absolute error as our loss function.

3.2.3 Single Image Iterative Model

We wanted to test a model that could be used iteratively to find the focus point. With each iteration we wanted to come closer to the target. Therefore it could not rely on images taken from a fixed position like our previous model. It needs to work on any distance. We decided that the model shall receive as inputs an image of the target, that image's focus score and its distance from the starting position. It will output the distance of the predicted focus point from the start position. The idea is that we can feed it an image to get a predicted focus point, move the lens to that position, take a new picture and repeat that process.

The data sets were created by choosing randomly 20 images from each run. With 72 runs we get 11520 total samples in our three data sets. This was to much data to load all at once in memory of our training machine, therefore we had to save the images as arrays on the disk using the pickle library and load them in dynamically during execution.

We tested the best way of analyzing the image, our candidates were convolutional networks and flattened dense layers. Then we experimented with the number and size of layers when connecting the image analysis model with the rest of the inputs. The training and validation loss was used as our guide during this testing. As our loss function we used mean squared error.

3.2.4 Single image Pretrained Model

With our final model we would try to replicate the model used in previous work on focusing microscopes [13]. This model would only take one image as input to a pretrained image classification network. We removed the top layer and flattened the output to then add our own output layer with one node using a linear activation function. The output represents the distance the lens would have to be moved to achieve focus.

Our data set was created by taking 50 random images from every run, this left us with a total of 28800 images. Similarly to the previous model we can not store that many images in our memory. We had to store them on disk the same way.

We experimented with different Keras built in pretrained models e.g. all the different mobilnet and efficientnet versions. For all models we experimented with different values on parameters like the width of the network. We also studied the performance impact of using different optimizers.

Chapter 4
Results

4.1 Performance

The best performance we could reach on the test data set using the three model concepts can all be viewed in Table 4.1. The tri-image model had by far the worst performance with an average of 122 µm. In second place we have the single image iterative model with an average error of approximately 25 µm but with a relatively high variance, the standard deviation was around 17 µm.

The best performing model by far was the single image pretrained one. It has the lowest mean error of 4 µm. As a point of reference we have included the bottom row called "always mean". It shows the error we would get if we where to choose the mean distance from the entire data set without taking any input.

Table 4.1: The performance of all three models. Included the error if a model were to always guess the mean distance of the data set.

Model	Mean Abs Error (µm)	Standard Deviation (µm)	Test Samples
Tri-Image model	112	30.1	115
Iterative Model	25.0	16.9	3090
Pretrained Model	3.70	5.27	5219
Always Mean	27.1	20.5	587

4.2 Tri-Image model

The resulted model can be viewed in Figure 4.1. The input images are given in 1D arrays because 2D representations of the images gave worse performance. This includes the pretrained imaging classification and convolutional networks. The less complex alternatives gave better performance.

Afterwards they are fed through dense layers before some are dropped out. These three chains gets concatenated into another series of dense layers before they reach the output layer.

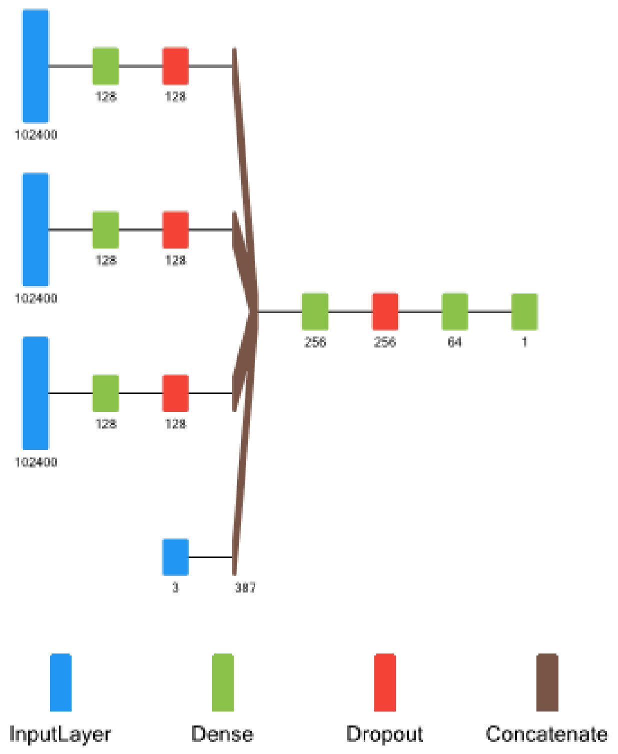

Figure 4.1: A visualization of the Tri-Image model.

In Figure 4.2 we can observe the training and validation mean absolute error in meters during the first 50 epochs when training the model.

4.3 Single Image Iterative Model

A figure of the resulted model (Figure 4.3) can be seen below. The sample image is fed through the first input layer as a 1D array and gets put through a max pooling layer. Its kernel size is 160, half the width of the image. Afterwards it gets fed through a couple of dense layers before doing a dropout. This image analysis gave the best performance compared to using 2D CNNs.

The focus score and distance from the start position get put through a dense layer and becomes concatenated together with the image. Everything is then fed through more smaller dense layers to the output layer.

The errors of the predicted distances in relation to the sample's distance to the focus point can be viewed in Figure 4.4. It shows that there is no relation between the distance to focus point and the accuracy of the prediction.

The mean squared error of the training and validation can be seen for all 50 epoch in Figure 4.5.

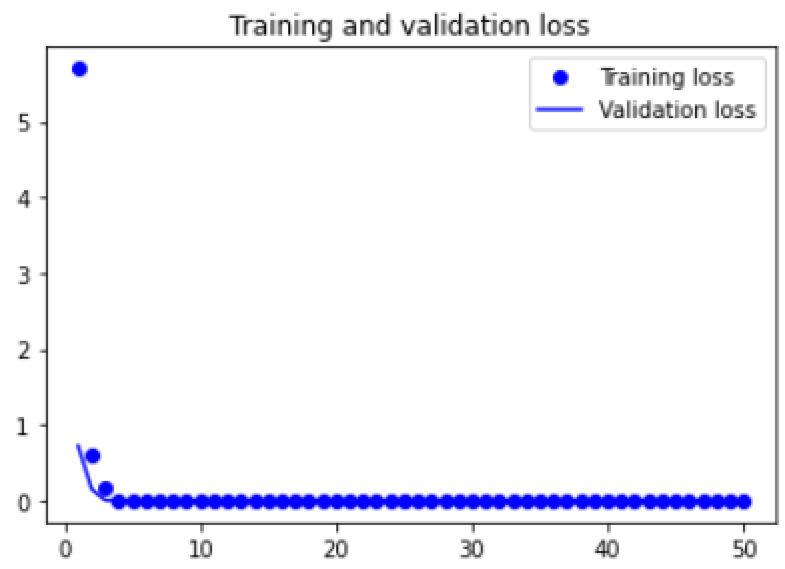

Figure 4.2: Training and validation mean absolute error in meters during the first 50 epochs.

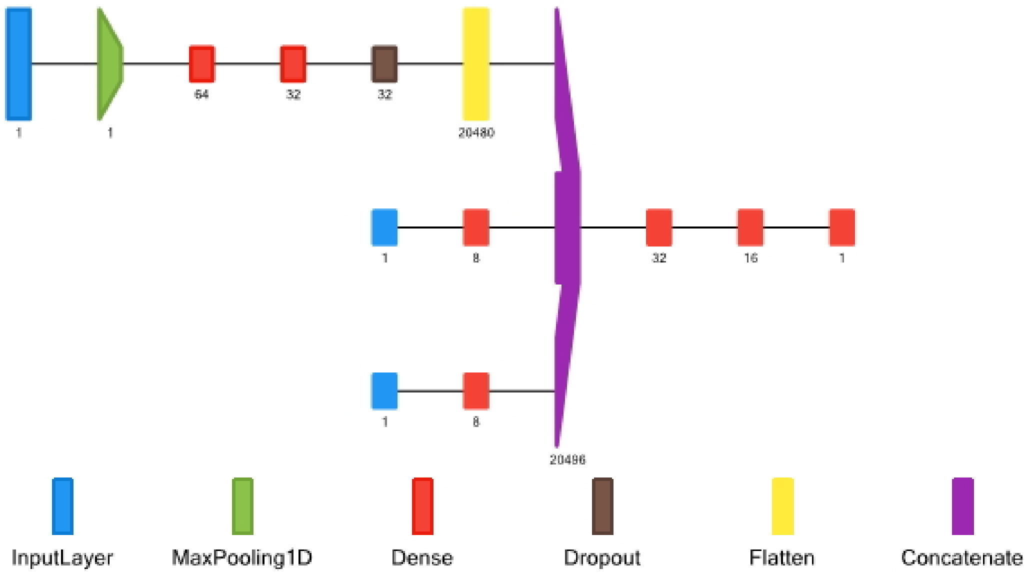

Figure 4.3: A visualization of the single image iterative model.

4.4 Single Image Pretrained Model

It is the best performing model of the three (Table 4.1). Of all pretrained networks built in Keras the MobileNetV2 gave the best result. A common theme for all the different pretrained models was that the more trainable parameters that were present the worse the validation score would be. By that logic the alpha parameter of the MobileNetV2 which represents the width of the network was set to its lowest setting of 0.35. After experiencing a very unstable validation score during training when using the adam optimizer we tried the stochastic gradient descent which eliminated the volatile validation. The learning rate had to be set to 0.0001 to prevent the error from blowing up in the first training batches.

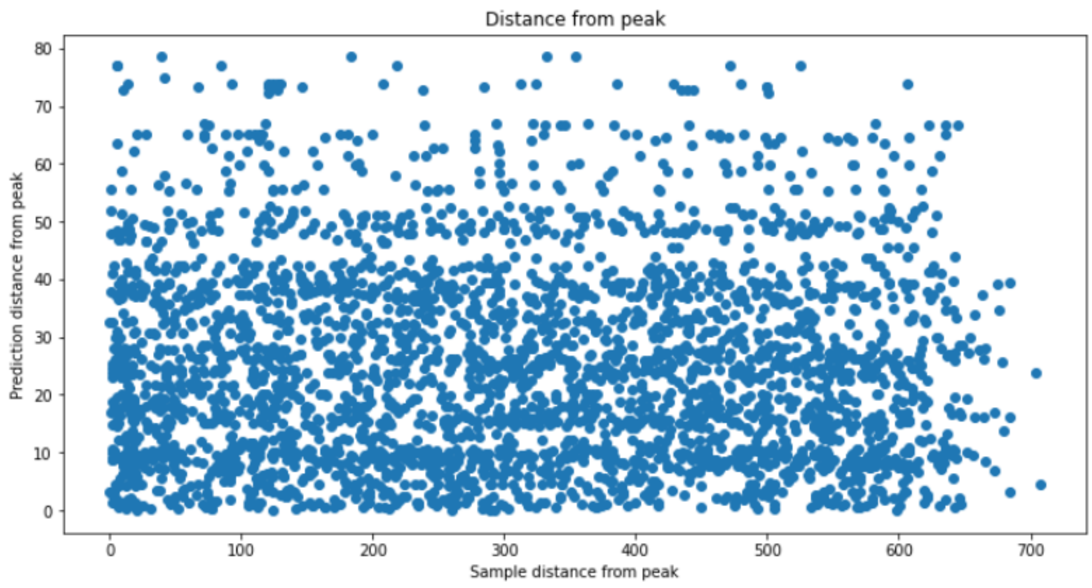

Figure 4.4: Scatter plot of the absolute error of the prediction on the test set in the y-axis. The x-axis represents the sample's distance from the correct focus point.

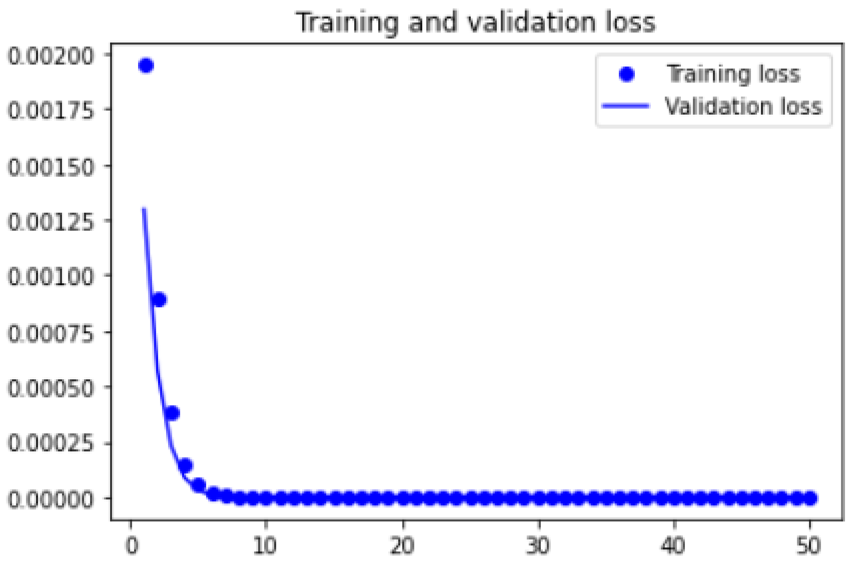

Figure 4.5: Training and validation mean squared error in meters during the first 50 epochs.

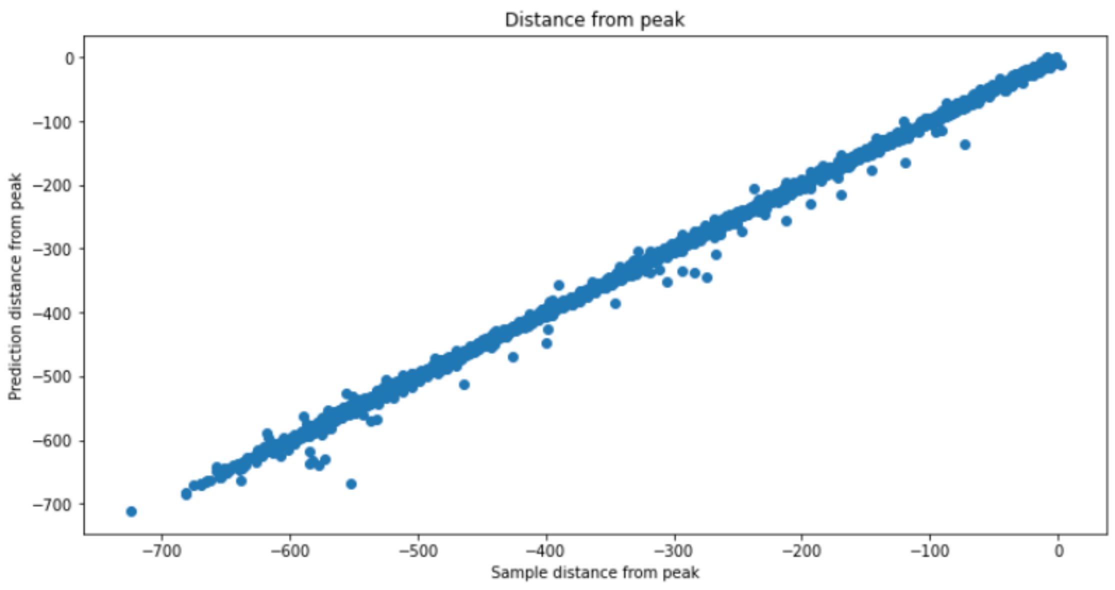

Figure 4.6: Scatter plot of the predictions made on the test set by the single image pretrained model on the y-axis with the samples distance from the peak in the x-axis.

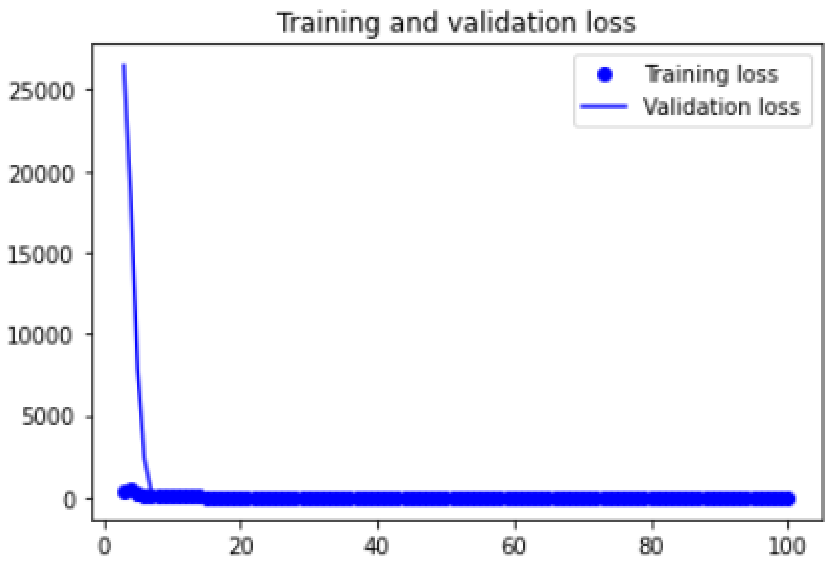

Figure 4.7: Training and validation mean squared error in meters during the first 300 epochs.

Chapter 5
Discussion

5.1 Performance

5.1.1 Tri-Image Model

Let us begin by analyzing each model, starting with the worst performer, the tri-image model. It had an mean absolute error of approximately 112 µm (Table 4.1). To put it into perspective of how bad it performed we compare it to if we were just guessing the mean focus distance regardless of input. It can be viewed in the same table. Using that method would only give an error of around 27 µm. One reason for this poor performance could be that we did not have as much training data compared to the next two models. This is because how of we chose this network to analyze the problem. We were using three images from the same position in all runs. In other words, of around 600 images per run we limit ourselves to three. This is a clear disadvantage the other two models does not have. But looking at Figure 4.2 we can see that the model did not seem to show any sign of too few data samples, like overfitting. Even with all of this taken into account, with performance this bad our conclusion is that this model architecture is not suitable for this task.

5.1.2 Single Image Iterative Model

Now for the single image iterative model which performed significantly better. The absolute mean error was around 25 µm. But if we put it into the context of guessing the mean value without taking input into account it only performs slightly better with a couple of micrometer smaller mean error and standard deviation.

The idea of the model was that it should work iteratively, where it would get closer to the focus point for each iteration. This would require the model to make better predictions the closer the input is to the focus peak. It seemed logical that this was the case, after all the target is a lot more visible close to the peak. But if we take a look at Figure 4.4 we can clearly see that distance to the peak and accuracy does not correlate with each other. This leaves us with a model we cannot iterate upon, that we can only run a single time and with only a slightly better performance than guessing the mean.

5.1.3 Single Image Pretrained Model

Our last model, the single image pretrained one was the absolute best performer by a wide margin. It had a absolute error of approximately 3.7 µm easily beating every other alternative in this study. Of all the pretrained models we tried it was the MobileNetV2 that performed the best. The same network used in the previous study about passive focus in microscopy [13]. Of all the MobileNet versions this was the one that could be scaled down the most with the alpha parameter. Making it the smallest network of all the Keras built in ones.

A big problem with the training of this network was the unstable validation error. With more trainable parameters came more fluctuating validation errors. This indicates that the bigger networks were too complex for our problem which often led to overfitting.

Another factor that played a big role was the choice of optimizer. When the optimizer was changed from adam to stochastic gradient descent we also saw the validation error stabilize between epochs. The learning rate needed to be lowered by a factor of 10, from 0.001 with adam to 0.0001 with SDG otherwise the training error would blow up after a couple of epochs. SDG is a much simpler optimizer then adam, continuing the trend that simpler methods provide better performance.

That does not mean that overfitting was completely eliminated. In Figure 4.6 we can observe how each sample in the test data set was predicted compared to the ground truth. It is evident that there exists a couple of outliers, the largest one being approximately 150 µm. Still, with the standard deviation being around 5 µm it seems that these outliers are few and far between.

Because our runs on the test machine during the data collection ended soon after the peak was found there is some bias in our data sets. During the first iterations of our model generation we realized that the model would never predict that the sample has already passed the focus peak. We conclude that it has to do with our biased data. The training sets contain too few samples that are taken from behind the peak to generate a big loss for the model to correct it.

There could be one more reason too. Just one image may not be enough information for a neural network to determine if the focus point is in the front or back. Some lenses used at Axis are symmetrical in that way. It is not possible to determine if some blur is foreground or background blur. One solution to remedy this is used in the same study about passive focus mentioned before [13]. Instead of only using a single image they subtracted two images a set distance from each other. While they cited the benefits of this where noise reduction it could also give a direction for the model to orient itself with.

5.2 Validity

Our biggest validity issue is in our opinion the diversity of our data set. As mentioned before we only used 36 unique lens-sensor combinations on two machines. It is one of Axis' most produced cameras. 36 combinations is not enough to represent such a mass produced product. If we had been able to use the data from production this would not have been a problem.

We mentioned before that there is no risk that our samples have outliers since all our camera modules have already passed quality control. That is in part a good thing, but it also poses a problem. If we are only training the network using parts that we know have no issues the network will not be able to handle faulty parts.

5.3 Future Work

The third model using a pretrained network seems like a good starting point for future experimentation. While already providing great performance there is room for improvement. Different forms of image pre-processing could be studied, for example, using a subtraction of images instead of a single image. Or a applying some blurring filter, which previous studies have found success with [13].

The symmetry problem needs to be studied more. Future studies will have to use the entire focal stack in their data sets, not stop right after the focus point has been found. If we were able to go back in time and change one thing it would be this. A small change that would increase the quality of the data immensely.

There were also some other ideas that where not able to be realized in this study, regarding the iterative model. We were not able to make it converge because it did not make any better predictions the closer it got to the focus point. The model is very simple for it cannot use the information gained from the previous iterations. Our ambition was to make a recurrent neural network (RNN), but it was a lot harder to implement in Keras than first believed. The data would have to be loaded dynamically since it would be wholly dependent on the output.

We have not discussed how big of an impact the fact that we used two testing machines has had on the data. In appendix B there is a histogram showing the different distributions the two testing machines produced. We can see that there is not a big difference the averages only differ in a couple of micrometers. Still this is only two machines, it is not enough to conclude that the difference is negligible. Perhaps a better performance could be achieved if a model was specific to each test machine.

Chapter 6
Conclusion

With this work in optimizing active alignment by developing a passive focus neural network we can draw multiple solutions.

First we created three ANN models based on three different concepts. The first was using images from the same position. Even when fine tuning and optimizing we where not able to able to make it perform nearly as well as we hoped. One reason could be the lack of data because of the fixed positions.

The second concept built on the assumption that a model that could be used at any position would be able to iterate upon its own results and eventually converge towards the focus point. This was not the case, proximity did not correlate with accuracy. Perhaps making the model recurrent would result in better performance.

Our last concept was the most successful one. We were inspired by previous work to use a pretrained network. This provided very good results with the mean error of only a few micrometers.

Did we mange to optimize the active alignment process? Yes we did. The original approach required the testing machine to take 20-30 images whereas the new method only needs one to come close.

There is still work to be done. The model produces still a couple of outliers, the validation loss is much higher then the training loss. It can be made more generalizable. Our opinion is that image pre-processing and smaller networks would be a good place to start.

References

[1] Abdullah Abuolaim, Abhijith Punnappurath, and Michael S. Brown. Revisiting autofocus for smartphone cameras. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[2] Chih-Yung Chen, Rey-Chue Hwang, and Yu-Ju Chen. A passive auto-focus camera control system. *Applied Soft Computing*, 10(1):296–303, 2010.

[3] Happiness Ugochi Dike, Yimin Zhou, Kranthi Kumar Deveerasetty, and Qingtian Wu. Unsupervised learning based on artificial neural network: A review. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 322–327, 2018.

[4] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. The MIT Press, 2017.

[6] Maximilian Hoeren, Daniel Zontar, Armin Tavakolian, Marvin Berger, Susanne Ehret, Temirlan Mussagaliyev, and Christian Brecher. Performance comparison between model-based and machine learning approaches for the automated active alignment of FAC-lenses. In Mark S. Zediker, editor, *High-Power Diode Laser Technology XVIII*, volume 11262, pages 60 – 69. International Society for Optics and Photonics, SPIE, 2020.

[7] http://netpbm.sourceforge.net/doc/pgm.html. Pgm.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[10] Adel Mellit, Massi Pavan, Emanuele Ogliari, S. Leva, and Vanni Lughi. Advanced methods for photovoltaic output power forecasting: A review. *Applied Sciences*, 10:487, 01 2020.

[11] Mattias Ohlsson and Patrik Edén. *Introduction to Artificial Neural Networks and Deep Learning*. Lunds Universitet, 2021.

[12] Henry Pinkard, Zachary Phillips, Arman Babakhani, Daniel A. Fletcher, and Laura Waller. Deep learning for single-shot autofocus microscopy. *Optica*, 6(6):794–797, Jun 2019.

[13] Tathagato Rai Dastidar. Automated focus distance estimation for digital microscopy using deep convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[14] samsung.com. How does auto focus work?

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] W.F. Schmidt, M.A. Kraaijveld, and R.P.W. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, 1992.

[17] trioptics.com. Trioptics products and applications.

[18] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

[19] Wengang Zhang, Hongrui Li, Li Yongqin, Hong Liu, Yumin Chen, and Xuanming Ding. Application of deep learning algorithms in geotechnical engineering: a short critical review. *Artificial Intelligence Review*, 54:1–41, 12 2021.

[20] Xiao Zhong and David Enke. Predicting the daily return direction of the stock market using hybrid machine learning algorithms. *Financial Innovation*, 5, 12 2019.

Appendices

Appendix A
Model Descriptions

A.1 Tri-Image model

```
Model: "model_22"
__________________________________________________________________________________________
 Layer (type)                Output Shape         Param #     Connected to
==========================================================================================
 input_24 (InputLayer)       [(None, 102400)]     0           []

 input_25 (InputLayer)       [(None, 102400)]     0           []

 input_26 (InputLayer)       [(None, 102400)]     0           []

 dense_44 (Dense)            (None, 128)          13107328    ['input_24[0][0]']

 dense_46 (Dense)            (None, 128)          13107328    ['input_25[0][0]']

 dense_48 (Dense)            (None, 128)          13107328    ['input_26[0][0]']

 dropout_18 (Dropout)        (None, 128)          0           ['dense_44[0][0]']

 dropout_19 (Dropout)        (None, 128)          0           ['dense_46[0][0]']

 dropout_20 (Dropout)        (None, 128)          0           ['dense_48[0][0]']

 input_27 (InputLayer)       [(None, 3)]          0           []

 concatenate_4 (Concatenate)   (None, 387)          0           ['dropout_18[0][0]',
                                                               'dropout_19[0][0]',
                                                               'dropout_20[0][0]',
                                                               'input_27[0][0]']

 dense_49 (Dense)            (None, 256)          99328       ['concatenate_4[0][0]']

 dropout_21 (Dropout)        (None, 256)          0           ['dense_49[0][0]']

 dense_50 (Dense)            (None, 64)           16448       ['dropout_21[0][0]']

 dense_51 (Dense)            (None, 1)            65          ['dense_50[0][0]']

==========================================================================================
Total params: 39,437,825
Trainable params: 39,437,825
Non-trainable params: 0
__________________________________________________________________________________________
```

A.2 Single Image Iterative Model

```
Model: "model_27"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 Input_Layer (InputLayer)       [(None, 102402, 1)]  0           []

 split_layer_17 (SplitLayer)    [(None, 102400, 1),  0           ['Input_Layer[0][0]']
                                 (None, 1, 1),
                                 (None, 1, 1)]

 max_pooling1d_16 (MaxPooling1D  (None, 640, 1)      0           ['split_layer_17[0][0]']
 )

 1First_dense (Dense)           (None, 640, 64)      128         ['max_pooling1d_16[0][0]']

 1second_dense (Dense)          (None, 640, 32)      2080        ['1First_dense[0][0]']

 dropout_37 (Dropout)           (None, 640, 32)      0           ['1second_dense[0][0]']

 2First_dense (Dense)           (None, 1, 8)         16          ['split_layer_17[0][1]']

 3First_dense (Dense)           (None, 1, 8)         16          ['split_layer_17[0][2]']

 flatten_48 (Flatten)           (None, 20480)        0           ['dropout_37[0][0]']

 flatten_49 (Flatten)           (None, 8)            0           ['2First_dense[0][0]']

 flatten_50 (Flatten)           (None, 8)            0           ['3First_dense[0][0]']

 concatenate_16 (Concatenate)   (None, 20496)        0           ['flatten_48[0][0]',
                                                                  'flatten_49[0][0]',
                                                                  'flatten_50[0][0]']

 c2_dense (Dense)               (None, 32)           655904      ['concatenate_16[0][0]']

 c3_dense (Dense)               (None, 16)           528         ['c2_dense[0][0]']

 last_dense (Dense)             (None, 1)            17          ['c3_dense[0][0]']

==========================================================================================
Total params: 658,689
Trainable params: 658,689
Non-trainable params: 0
__________________________________________________________________________________________
```

A.3 Single image Pretrained Model

```
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_25 (InputLayer)       [(None, 320, 320, 3)]     0

 efficientnet-b0 (Functional  (None, 10, 10, 1280)     4049564
 )

 dense_24 (Dense)            (None, 10, 10, 64)        81984

 dense_25 (Dense)            (None, 10, 10, 1)         65

=================================================================
Total params: 4,131,613
Trainable params: 4,089,597
Non-trainable params: 42,016
_________________________________________________________________
```

Appendix B
Data Analysis

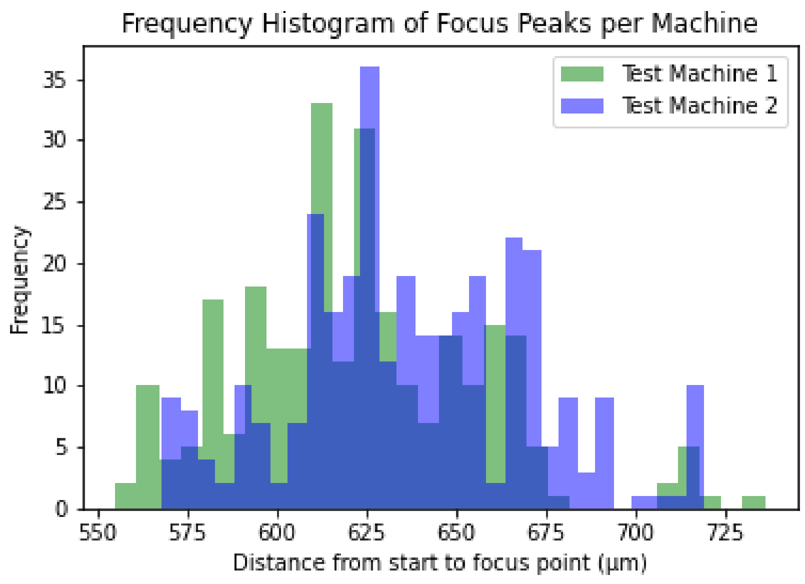

Figure B.1: The distribution of the focus peaks by the two different testing machines.

Table B.1: The mean and standard deviation of the distribution of the focus peaks by the two different testing machines.

Machine	Mean (µm)	Standard Deviation (µm)
Machine 1	623	33.2
Machine 2	638	33.1

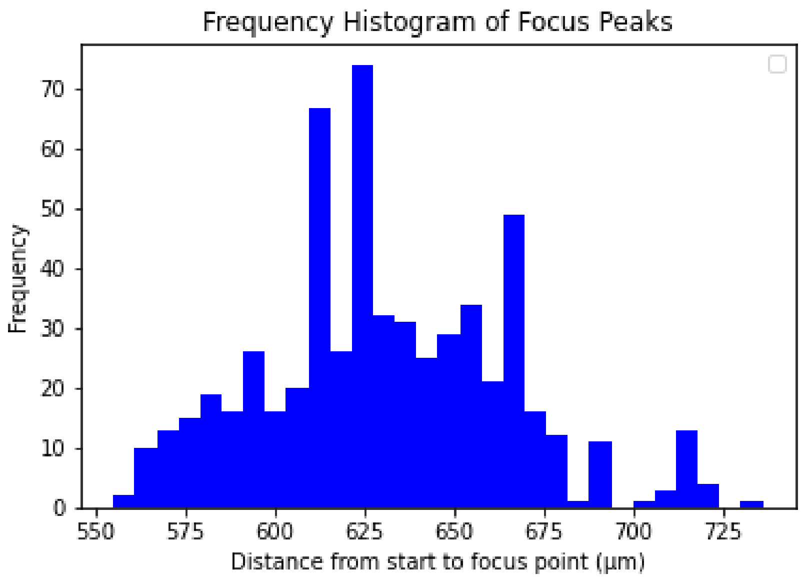

Figure B.2: The distribution of the focus peaks.

Table B.2: The mean and standard deviation of the distribution of the focus peaks.

Mean (µm)	Standard Deviation (µm)	
Peaks	631	34.0

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-06-20

EXAMENSARBETE Optimizing active alignment during assembly using neural networks
STUDENT Dennis Jusufovic
HANDLEDARE Dr. Dennis Medved (LTH), Martin Hellaeus (Axis), Dr. Stefan Plogmaker (Axis)
EXAMINATOR Prof. Jacek Malec (LTH)

Optimering av active alignment med deep learning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Dennis Jusufovic**

Med den ständigt ökande tillgängligheten av maskininlärning börjar företag tänka om sina tidigare lösningar. Även på Axis prövas gamla idéer på nytt. Vi undersöker om det går att optimera ett steg i produktionen kallat active alignment med hjälp av neurala nätverk.

Att bygga kameror är ett väldigt precist arbete. Något som skiljer sig på micrometern kan leda till en kamera med mycket sämre fokus. I den här artikeln fokuserar vi just på när en linsmodul och ett sensorkort ska föras samman. Det är viktigt att de har rätt avstånd och lutning från varandra. Det krävs dock något mer än precision för den här uppgiften, eftersom varje sensorkort och linsmodul är lite annorlunda. Med active alignment tar man dessa variationer i beaktning, man analyserar bilder tagna med sensorn och räknar ut den optimala positionen och lutningen. Detta gör att det går att använda billigare delar men ändå ha en bra kamera.

Det finns ett specifikt steg i active alignment kallat coarse alignment. En linsmodul förs steg för steg mot sensorn medan den tar bild efter bild. Bilderna analyseras och fokus uppskattas. När datorn detekterar en fokus-topp har den hittat det bästa avståndet. Målet med den här studien är att hitta en effektivare sätt att utföra detta på. Någon modell som kan förutsäga var fokus-toppen kan finnas istället för att leta blint efter den. Vi utvecklade tre stycken neurala nätverk som potentiella kandidater att ersätta den gamla metoden.

Resultatet visar att en av modellerna verkar lovande. Den bygger på ett redan förtränat nätverk

kallat MobileNetV2 och behöver endast en bild av sensorn för att göra en gissning. Den kan utföras snabbare och med mindre bilder än orginalmetoden och överträffar de andra två modellerna. Den är inte helt perfekt, det finns fortfarande några avvikelser med testdatan, men även med det i åtanke så kan den här metoden ses som ett framsteg.