University of Southern Denmark

Bachelor Thesis, (f20)

Civil Engineer in Robot Technology - 6th semester

Deadline for submission: June 2nd 2020

Keypoint Detection using Computer Vision and Artificial Intelligence

Student:
Jacob Falgren Christensen
jacoc17@student.sdu.dk
25/06-1997
Eks.nr.: 113285149

Student:
Troels Zink Kristensen
tkris17@student.sdu.dk
30/04-1997
Eks.nr.: 455427

SDU Advisor:
Anders Glent Buch
anbu@mmmi.sdu.dk

Inwatec Advisor:
Tudor Morar
tudor.morar@inwatec.dk

Abstract

The objective of this bachelor thesis is to determine the viability of using deep learning and computer vision to predict pick points that are usable by an Inwatec separator-robot. This combination of deep learning and computer vision must facilitate the design and implementation of a convolutional neural network as well as the preprocessing of the training and test images. Images used for training and testing are gathered on the Inwatec separator-robot, using a data acquisition-program, sampling random successful pick attempts within the allowed pick area.

To preprocess the images, OpenCV is used. The images are first processed and then split into the training set and the test set. The images are scaled and normalised to fit the input of the CNN.

Using the TensorFlow Keras API, a simple CNN used for regression is designed and trained. Once the model is trained, it is saved and ready for implementation on the separator-robot. This implementation is done using the TensorFlow Serving API. Using this API, the saved model runs in a Docker container, and listens for a specific port waiting for new requests. These requests are sent to the model by first generating a JSON-file from the values in an image, and then using an HTTP-client, written in C++, to add the necessary headers. When the headers are added, the message is ready for transmission. Once the model has received the message and processed the image, a response is returned containing a pick point that can be used by the separator-robot.

Once trained and implemented, the models are tested on the separator-robot. The purpose of these tests is to examine the success rate of the models and compare them to the original separator-program. The tests show a higher success rate in the models compared to the acquisition-program. However, the models still perform slightly worse than the original separator-program.

Preface

This bachelor thesis is composed by Jacob Falgren Christensen and Troels Zink Kristensen.

The project is associated with studies in BSc in Engineering (Robot Systems) on the 6th semester in spring 2020 at the University of Southern Denmark in Odense. The project covers 15 ECTS-points.

The project started February 3rd with the approval of the project description, and was concluded on the 2nd of June with the submission of this report.

We want to thank our advisor at SDU, Anders Glent Buch, for guiding and helping us. We also want to thank Inwatec and our advisor at Inwatec, Tudor Morar, for giving us the opportunity to work closely with a company and work with a real-life problem.

Reading guide:

Literature references are shown by $^{[number]}$ and refers to the reference list in section 11. Some terms within the report are at first referred to using the full name; definitions and alternate ways of referring to the term can then be found in the footnotes. Code specific expressions are formatted using the `menlo font`. An overview of the electronic appendix can be found in section 12 Appendix. Figures used in the report can be found in original size in the folder "Figures" in the electronic appendix.
The PDF is interactive, which means that it is possible to click on references to chapters, figures, equations and code listings to jump directly to them.

Table of Contents

1 Introduction

The purpose of this project is to create a deep neural network that will predict keypoints[1], and send it to the Inwatec separator-robot[2] that has to pick-up one piece of laundry/garment at a time. The neural network receives an image for every input, in which the x- and y-values are extracted from and used as labels/ground truth. All these images go trough preprocessing, which includes splitting the data into a training set and a test set. The output of the network will then be a predicted (x,y)-coordinate for a specific image that can be fed to the robot, which will attempt to pick up a piece of laundry at that location.

A neural network is constructed with different properties, such as activation functions, neurons, filters, batch-size, and so on. A neural network consists of several layers; the first one, the input layer with an input shape corresponding to the image size; several hidden layers; and an output layer that delivers the wanted output based on the purpose of the network. The input is mostly made of images, text or sound, and with enough data, the network will be able to predict what it thinks is the best possible outcome.

The robot is called THOR and is categorised as a separator. The existing program on the robot will be referenced to as the separator-program doing the report. THOR is shown below in figure 1. The image shows four essential parts of the robot: the bin surrounding the region of interest that holds all the garments; the 3D-camera that captures all the images; the delta robot, which picks the garment from the bin; and the delivery arm that grabs the garment from the delta robot and puts it onto the delivery conveyor belt, which eventually will result in a successful delivery.

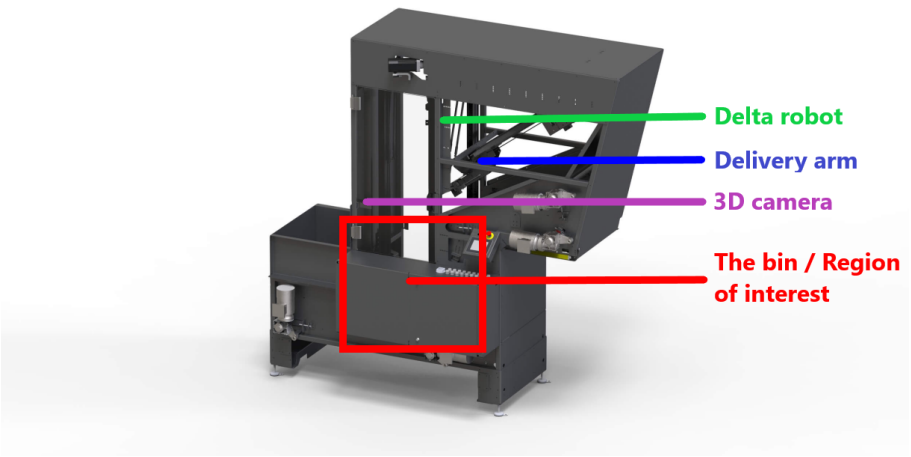


Figure 1: *THOR - Separator-robot.*[4]

[1] 2D-coordinate on an image. Also referred to as pick points.
[2] Referred to as robot.

2 Problem Description

Inwatec has a robot whose purpose is to pick pieces of laundry such as towels and garments one by one from bulk, in order to feed them to a system that identifies and sorts them according to their washing category. A program is required to help identify specific keypoints that would allow the robot optimal chances of a successful pick. A 3D-camera is used to capture an image of the laundry. This image is then evaluated by a deep neural network. The network is designed to extract keypoints from the image.

Purpose

Construct a deep neural network using the convolutional neural network (CNN) architecture to detect and extract keypoints in a given image. Using these keypoints, the performance of the robot is compared to the existing separator-program. The implemented models of the neural networks have to be implemented into the separator-program.

Sub-Questions

- Data Acquisition and Preprocessing
  - What type of data is needed?
  - How much data is needed to achieve a good model fit?
  - What kind of preprocessing should be implemented?
- Design of Network
  - What is the best way to construct a CNN for the given problem?
- Training and Test of Network
  - What is the best way to train and test the network?
  - How can the hyperparameters be tuned to improve performance?
  - How can the performance of the network be quantised?
- Analysis of Model Performance
  - How does the model perform compared to the separator-program?
  - Which model is the best one?

3 Data Acquisition and Preprocessing

Data acquisition is a subject that is very essential for this project. Without data, a neural network cannot be used. Once the data is acquired, preprocessing can be applied to perform various tasks to achieve the best possible outcome.

3.1 Data Acquisition

The data is acquired from an existing program at Inwatec referred to as the separator-program. The separator-program is based on a traditional computer vision approach, in which pick points are detected through a 3D-camera. The images captured by the camera are used to find these pick points by looking at peaks, colour, and other patterns. Based on this image analysis, the most promising pick point is selected. The pick attempts are graded on a scale from 0 to 3, with 0 being a failure and 3 being a successful delivery of a garment. The scores 1 and 2 are considered as partial successes, which only yields a successful pick attempt, but not a successful delivery.

To ensure the neural network learns a new behaviour compared to the original separator-program, the generation of pick points is changed to generate pick points randomly within the allowed pick area. The successful deliveries using these points are then saved; ready for preprocessing. The pick attempts with a score below 3 are ignored, as failures and partial successes are not wanted within the training data. The images are saved with a specific name to ensure dissimilarity. An example of a name is:

- 3 - Colour - X349 Y188 - 137.jpg
- 3 - Depth - X349 Y188 - 137.jpg

The name represents the score, type of image, x- and y-positions, and the image number. Both a colour (RGB) and a depth image are saved for each pick attempt.

3.2 Preprocessing

The images go through preprocessing to allow the use of a large network and more data. After the images are loaded into the program, the (x,y)-coordinate is extracted from the file-name, and used as the ground truth[3].
All of the images have the following dimensions: 848×480 pixels. The colour images have three channels, which results in $848 \times 480 \times 3 = 1,221,120$ values for each image. As the hardware used to train the models is limited in memory (see listing 1), this number of values will severely limit the size of the network.

[3] Also referred to as labels.

Therefore, the images have to be resized. The chosen dimensions of the resized image are a quarter of the original: 212×120 pixels. Figure 2 shows an example of an acquired image.

```
System 1:
- Processor      : i7-6700
- GPU            : GTX 1070ti
- System Memory  : 32GB

System 2:
- Processor      : i7-7700HQ
- GPU            : GTX 1060
- System Memory  : 16GB
```

Listing 1: *The used systems.*

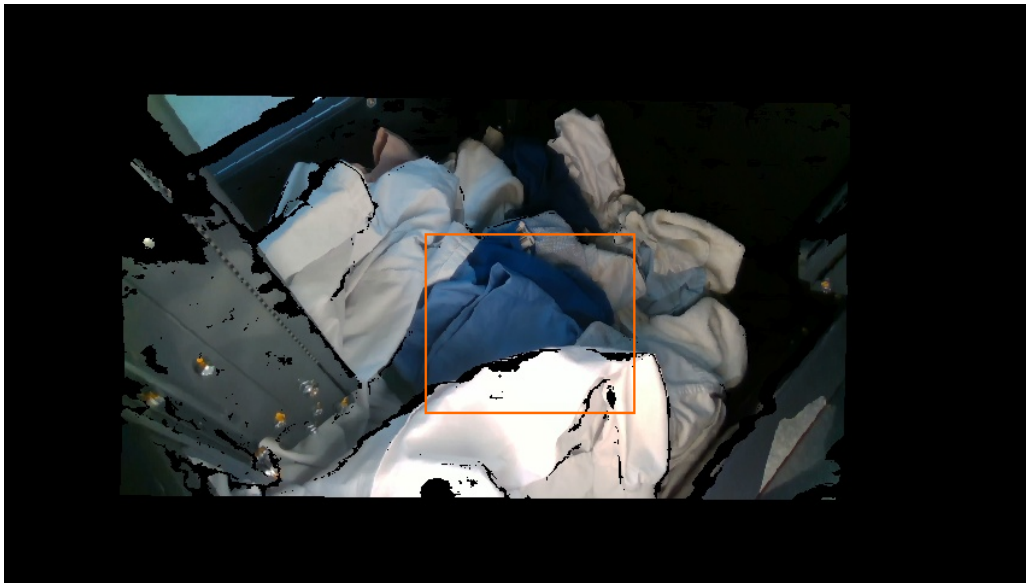


Figure 2: *An example of an image - Dimensions:* 848×480.

The image shows the bin with the region of interest (ROI), the orange square, with garments at the bottom. The ROI contains the pick area, which is the area the robot can pick from. The pick area changes for each image.

After an image is resized, all pixel values are normalised between zero and one. All labels are normalised to ensure that even though the image has been resized, the label still refer to the same location in the image. Each label is normalised using the following equation:

$$\text{Label}_{norm}[x] = \frac{\text{Label}[x]}{848.0}, \qquad \text{Label}_{norm}[y] = \frac{\text{Label}[y]}{480.0} \tag{1}$$

3.3 Partial Conclusion

10,000 colour and depth images are acquired from the robot. All of these images are scaled down to 25% to allow for a large network structure and large data quantity. All of the values in the images are normalised as well before training the network.

4 Design

In this section, the design of the neural network will be explained. As mentioned previously a CNN is used. CNNs specialise in feature extraction from images, thus this type of neural network is ideal to fulfil the purpose of this project. These features can then be used to predict the (x,y)-coordinate that is the best candidate for a pick point. In this section it is explained how a network, referred to as the base network, is designed.

4.1 Regression Neural Network

As the purpose of the base network is to predict an (x,y)-coordinate in a given image, the neural network needs to be tailored to this specific task. To accomplish this goal, a regression network can be used. This type of network learns the correlation between the data points it is training on and the ground truth. Instead of predicting a specific class, a regression network is able to predict one or multiple numbers such as coordinates. These coordinates would correspond to the pick point, which the CNN thinks is the closest match to the data it has trained on. Figure 3 shows an illustration of the used regression.

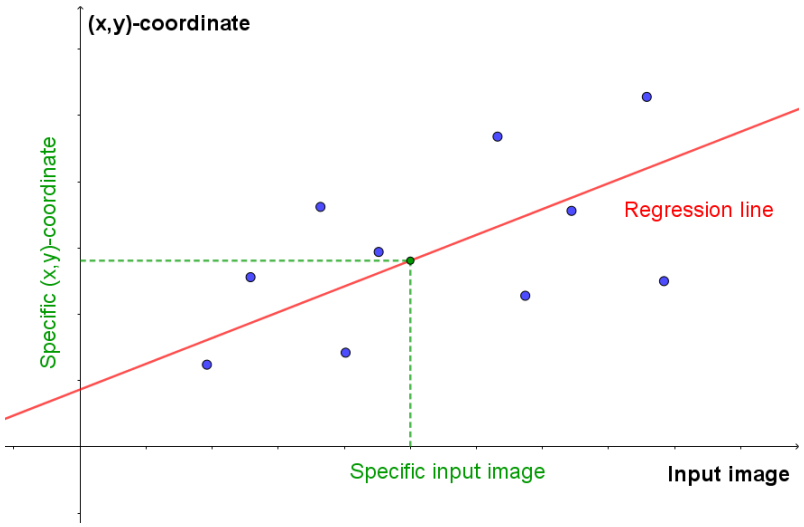


Figure 3: *Illustration of the used regression.*

4.2 Network Structure

This CNN has been designed using the TensorFlow Keras API, which allows for simple training and modification. Since the CNN is designed using Keras, several building blocks can be used. The ones used to design this CNN are shown in the list below:

- Convolutional 2D-layer
- MaxPooling 2D-layer
- Dense or fully-connected layer
- Dropout layer

The base network is designed to have a simple structure. This is done to test if a neural network is able to provide usable coordinates. A visualisation of the structure can be seen in figure 4. Using the aforementioned layers, a downsampling CNN is designed, consisting of a series of convolutional layers followed by pooling layers, and ending in a set of fully-connected layers.

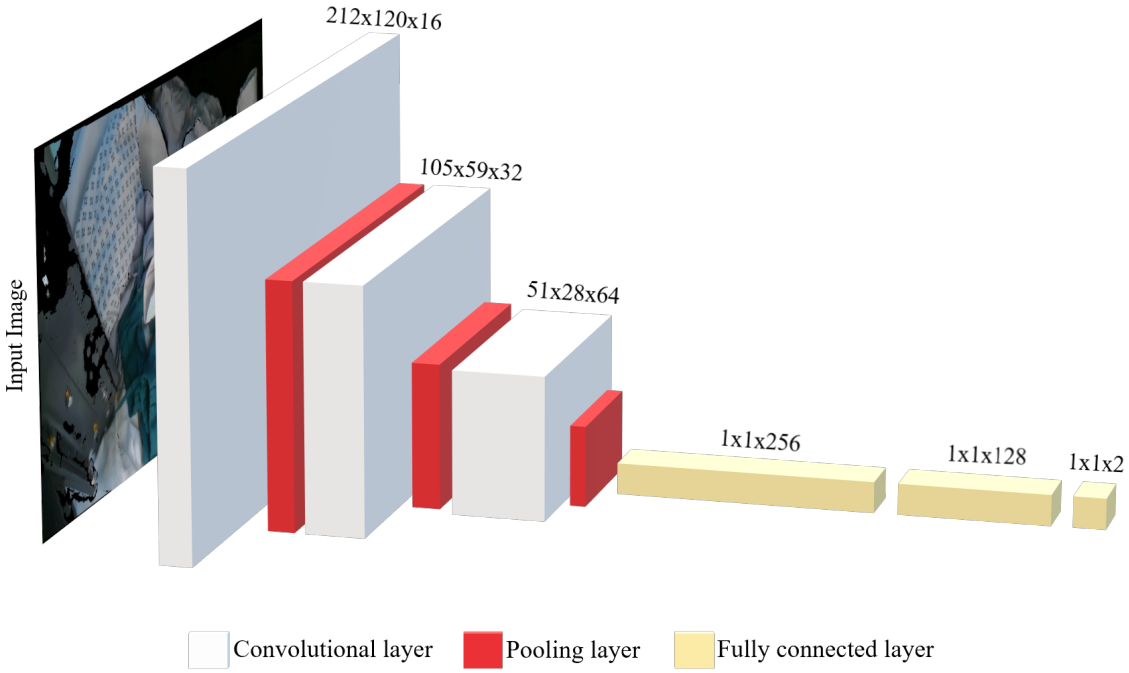


Figure 4: *Visualisation of the network structure.*

The network is structured using blocks to downsample the data, and fully-connected layers to predict the output. Each block consists of two layers. The first block contains the input layer; this layer works directly on the input image.

The first layer in a block is a convolutional layer and consists of a number of filters. Each filter will convolve over the image providing an activation map; these activation maps are then stacked and sent to the next layer. Activation maps represent the features the network extract with each filter, and the weights of the layer are adjusted during training to improve the quality of the features extracted.

The second layer of the block is a pooling layer; this layer pools the information contained in the activation maps, thus giving a smaller and more concise representation of the features. The reduction in size also helps conserve memory and allows for more filters to be used. Once pooling has been applied, the activation maps are sent to the next layer. This combination of one convolutional layer and one pooling layer is repeated twice.

The last part of the network is the fully-connected layers. These layers take the information provided by the activation maps, finding the relation between the features contained in the activation maps and the output of the layer. This simple network contains two fully-connected layers with a decreasing size to get more concise information. The network ends in a fully-connected layer acting as an output layer with the same number of neurons as an (x,y)-coordinate, and it is the value of these neurons that are the output of the network.

Activation Function

To calculate the output of each neuron in a layer, an activation function is used. Throughout this network, the Rectified Linear Unit (ReLU) activation function is used. The ReLU activation function can be seen in equation 2.

$$R(z) = max(0, z) \tag{2}$$

Where z is the value from the current neuron. ReLU is used because the calculations made during the ReLU-operation are faster than other options, thus decreasing training time. Due to the fact that the ReLU activation function does not saturate, this alleviates the vanishing gradient problem and allows the network to continue training at a high speed. A visualisation of the ReLU activation function can be seen in figure 5.

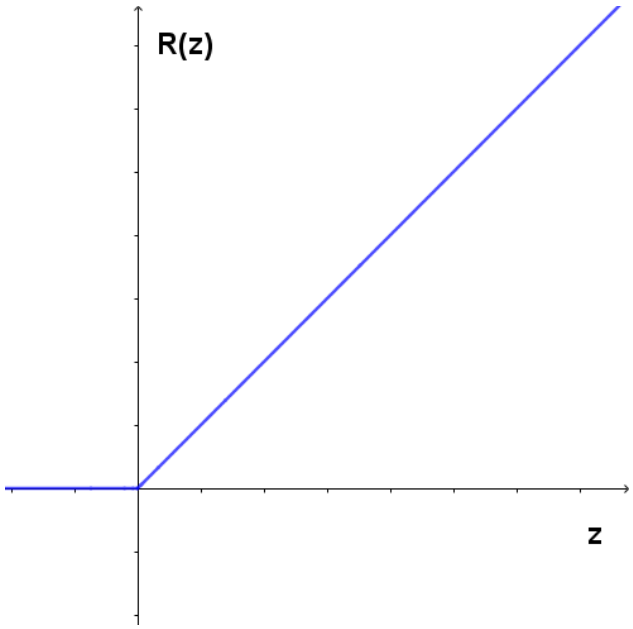


Figure 5: *Visualisation of the ReLU activation function.*

The only layer that is not using the ReLU activation function is the output layer. This layer uses the linear activation function to enable the network to perform linear regression.

Optimiser

The optimiser is used to adjust the weights during training. The Adam optimiser is chosen. This optimiser is chosen due to the factors listed below:

- Easy to implement
- Efficient computations
- Overall robustness

The Adam has proven to be one of the best optimisers when used on a CNN[3], which can be seen in figure 6.

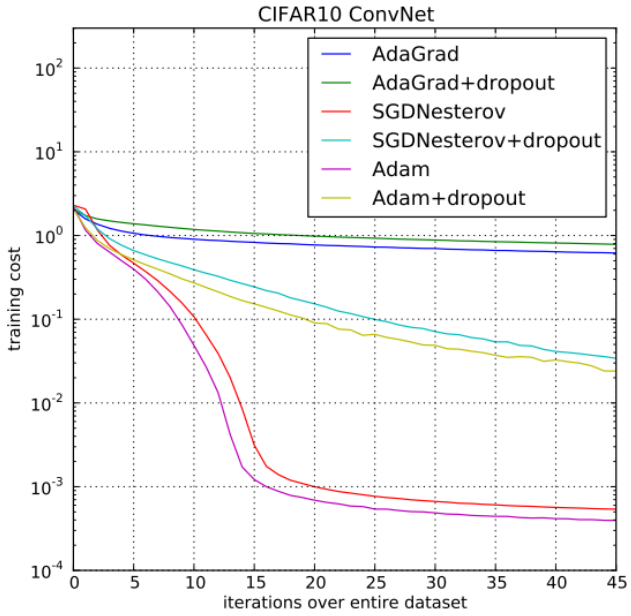
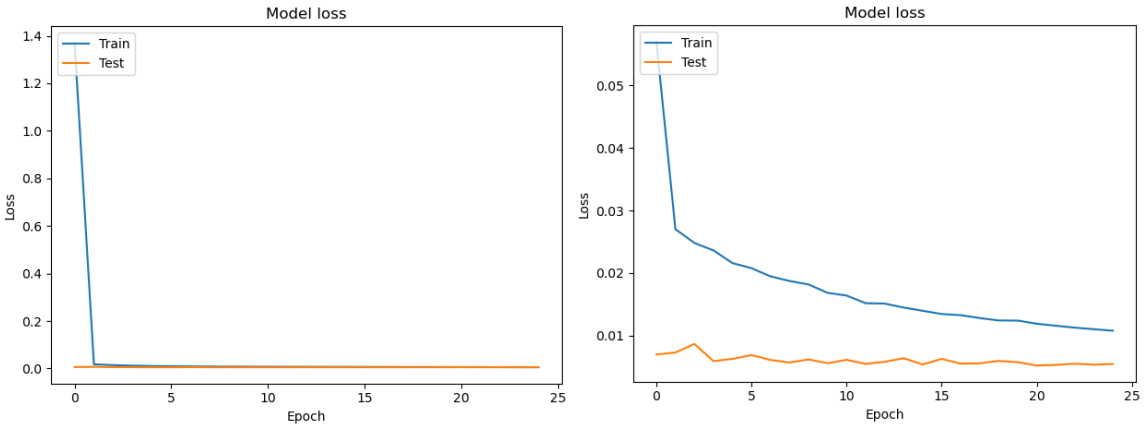


Figure 6: *Adam loss curve when used on a CNN with the CIFAR10 dataset.*

As can be seen in the the figure, the Adam optimiser achieves the lowest loss out of all the optimisers tested. Furthermore, the Adam optimiser shows a fast convergence.

The learning rate used by Adam is a hyperparameter that can be tuned to improve the performance of the model. When training with a default learning rate[1] of $\alpha = 0.001$, the corresponding loss curve can be seen in figure 7a. This curve is not desirable as it converges towards zero too fast. When the learning rate is reduced to $\alpha = 0.00001$, the training yields a better loss curve, which can be seen in figure 7b.



(a) *Loss curve using* $\alpha = 0.001$. (b) *Loss curve using* $\alpha = 0.00001$.

Figure 7: *Loss curves when using the Adam optimiser.*

Loss Function

A loss function is used to calculate the loss used for backpropagation. For this network, the Mean Squared Error (MSE) loss function is used. This loss function is very well-suited for regression tasks, as it penalises large errors much more than small errors. The equation of the MSE loss function can be seen in equation 3.

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y - \hat{y}_i)^2 \tag{3}$$

Where y is the ground truth value and $\hat{y}$ is the predicted value.

4.3 Implementation

The base network is implemented using the TensorFlow Keras API in Python. This is done using a Keras Sequential model, and a code example can be seen in listing 2. The `model.add(...)`-function is used to add new layers to the model. Once all the layers have been added, the model can be compiled and is then ready for training and testing.
When adding a layer to the model such as a `Conv2D`-layer, the amount of filters are specified, as well as the kernel-size and the activation function. This is the case for all `Conv2D`-layers. However, the first layer has an additional parameter, which is the shape of the input. `MaxPooling2D`-layers are added after each `Conv2D`-layer to downsample the data; these layers take a pooling size as a parameter, which decides the strength of the downsampling. The `Flatten`-layer is added after the last pooling layer to stretch out the activation maps to a vector containing all the values of the activation maps, which allows it to be linked to the fully-connected layer. `Dropout`-layers are added between the fully-connected layers to reduce overfitting.

```
model = Sequential()
model.add(Conv2D(16, kernel_size=3, activation='relu', input_shape=inputShape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Dropout(0.50))
model.add(Dense(128))
model.add(Dropout(0.50))
model.add(Dense(2, activation='linear'))
```

Listing 2: *The base model.*

An instance of the optimiser is declared when the model is compiled; afterwards the `model.compile(...)`-function can be used. A code example can be seen in listing 3.

```
Adam = Adam(learning_rate=0.00001)
model.compile(optimizer=Adam, loss='mse', metrics=['accuracy'])
```

Listing 3: *Model compiler.*

The optimiser is declared using the learning rate as the parameter. This instance of the optimiser is then fed to the `model.compile(...)`-function along with the chosen loss function and a variable telling the network which metric to track.

4.4 Partial Conclusion

In this section, it has been described how a simple convolutional neural network has been designed and compiled using the Keras API. It is now possible for the network to train on the provided data. The model is also capable of further improvements and can be easily modified.

5 Training

The neural network is now designed to perform regression. The input can be modified to either be singular or consist of multiple image types; that being colour, depth, or a combination of both. The base network is used as a foundation, where further modifications can be implemented. The modifications of the network and the testing hereof, will be explored in this section.

Contrary to a standard neural network, where the output is a specific class, the output of a regression network in this implementation is a 2D pick point (x,y-coordinate). As multiple points could yield a result that contains the same features as the ground truth, it is difficult to ascertain the quality of the prediction. To find a metric that the network performance can be evaluated by, the distance between the prediction and the ground truth is calculated. The hypothesis is that the distance should be as low as possible, that is, the predicted pick point should be as close to the ground truth as possible, since the ground truth is a good pick point, as the robot made a successful delivery during data acquisition.

The aim of the training process is to lower the average distance between all the predicted pick points and the corresponding ground truths.

Figure 8 shows the ground truth for this specific image at $(x, y) = (381, 191)$ marked as a blue dot. The predicted pick points from two different models are shown at $(x, y) = (403, 209)$ and $(x, y) = (440, 240)$ marked as red dots. It shows that the models do not necessarily predict a pick point right next to the ground truth.

Figure 8: *Ground truth and the predicted pick points from two different models.*

5.1 Tuning of Hyperparameters

A hyperparameter is a variable that determines the network structure and training of the network. Hyperparameters will be configured/tuned to see how they affect the performance of the network. The hyperparameters that will be tuned are:

- Number of images
- Number of epochs
- Batch-size
- Activation function
- Loss function

As more images yield better performance, the highest possible number of images were collected within the time frame. However, datasets containing a smaller number of images are tested as well to examine the performance of these.

The number of epochs is tuned to figure out, when the network overfits. A large number of epochs is desirable; however, the network should not overfit. Nevertheless, a larger number of epochs will also result in a higher training time.

A batch is the number of images processed by the network before the weights are updated. A batch-size of 16 means that 16 images are trained upon, and afterwards the weights are updated. One epoch with a batch-size of 16 would result in:

$$\#\text{updates} = \frac{\#\text{images}}{\text{batch-size}} = \frac{10,000}{16} = 625 \tag{4}$$

After one batch is processed, a loss is calculated, and based on this loss the model is improved after each batch.[2] A higher batch-size will lead to a faster training time, but information will be lost as well.

The hyperparameters are tuned individually with the base network as standard values. The standard values are:

- 10,000 images, each 25% scale equal to 212×120 pixels instead of 848×480
- 25 epochs
- Batch-size equal to 16
- ReLU activation function for the input layer and the hidden layers, and linear activation function for the output layer
- Mean squared error (MSE) as the loss function

Due to hardware limitations the image scale was capped at 25%. The hardware used is seen in listing 1. The aforementioned hyperparameters will be tested with 3-5 different values, which are shown in table 1:

Table 1: *The chosen values for each hyperparameter.*

	Values				
Images	100	1000	2000	5000	10000
Epochs	5	25	50	100	250
Batch-Size	4	8	16	32	128
Activation Function	ReLU	Sigmoid	Softmax	Tanh	–
Loss Function	MSE	MAE	Hubor	–	–

5.2 Results

All of the tests are shown below in table 2. The tests are executed with four different image types, which are colour, depth, normal maps, and a combination of colour and depth. The combination of colour and depth is two separate functional networks[7] concatenated before the first dense-layer (see listing 2). All of the black values in the table are the average distances between the ground truths and the predicted pick points. The average distance is in pixels. All results along with graphs can be seen in *Journal for Training* in the *Electronic Appendix*.

Normal mapping is a way to enhance details in an image. It is generally done using the unit vector from shading point to light source, and the unit vector normal to that surface. These two vectors are dotted with each other. In this way, the shape of the surface is approximated with three channels just as a colour-image. However, each channel represents an x-, y- and z-coordinate, respectively, instead of red, blue and green.[5] Normal mapping is configured on the depth images to figure out if there is a relation between the images and the ground truth. Figure 9 shows the corresponding normal map for the colour image in figure 8.

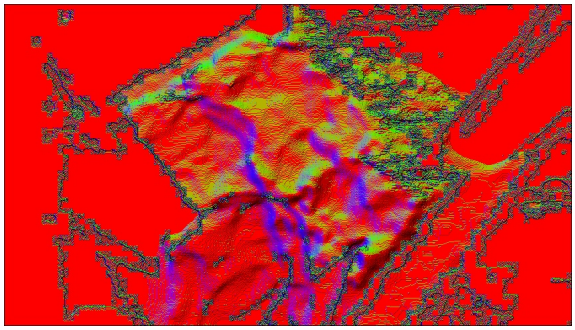


Figure 9: *An example of a normal map.*

Table 2: *The results from the different tests. The left column contains the image types trained upon, and the values in the middle are the average distances in pixels.*

Number of Images					
	100	1000	2000	5000	10000
Colour	67.5	63.5	60.0	58.1	56.3
Depth	54.6	59.7	57.5	58.3	57.2
Normal Maps	85.0	65.4	62.9	57.8	62.6
Colour + Depth	102.1	62.8	60.7	57.1	63.4
Number of Epochs					
	5	25	50	100	250
Colour	59.1	58.4	62.9	69.1	73.6
Depth	58.1	55.7	60.9	66.3	71.1
Normal Maps	56.1	57.5	64.1	68.3	67.6
Colour + Depth	58.6	58.8	63.2	66.0	65.6
Batch-Size					
	4	8	16	32	128
Colour	64.1	60.1	57.8	57.8	58.4
Depth	62.3	58.2	55.3	56.0	58.5
Normal Maps	63.2	61.2	58.0	60.1	56.2
Colour + Depth	63.1	61.7	59.4	56.6	56.5
Activation Function					
	ReLU	Sigmoid	Softmax	Tanh	–
Colour	57.9	59.4	62.4	58.6	
Depth	56.7	87.4	55.4	59.9	
Normal Maps	57.5	80.3	55.4	66.0	
Colour + Depth	60.9	64.3	55.8	61.3	
Loss Function					
	MSE	MAE	Huber	–	–
Colour	58.1	58.5	58.0		
Depth	57.8	55.8	56.2		
Normal Maps	62.9	56.7	57.1		
Colour + Depth	58.2	57.9	58.4		

From table 2 the following can be concluded:

Number of Images

In general, a larger number of images results in a lower average distance. There are some exceptions as seen in the table. However, it seems that 5000 images could have been enough, as all four types of images are below 60 pixels at this number. Furthermore, the depth type has the lowest values overall, and the values of the colour type decrease as more images are used.

– Colour and depth are the best candidates at 10,000 images.

Number of Epochs

From the table, it can be seen that a larger number of epochs yields a larger average distance, except from 5 to 25, where the values are about the same. The best image types are depth and normal maps at both 5 and 25 epochs. However, the difference is not significant.

– Depth and normal maps are the best candidates at 25 epochs.

Batch-size

A larger batch-size seems to fit well for normal maps and colour+depth. Nevertheless, a batch-size of 16 seems to fit well for colour and depth. Overall, a small batch-size such as 4 and 8 results in a larger average distance.

– As a smaller batch-size keeps more information about the images compared to a larger one, colour and depth are the best candidates at a batch-size of 16.

Activation Function

The ReLU and the Softmax activation function have the overall lowest average distance for nearly all image types. The Tanh activation function is close to the other two, and the Sigmoid is quite bad. However, as mentioned in section 4 about the activation functions, the ReLU is the best one in this type of network.

– All of the tested activation functions except Sigmoid are close to each other in performance, but ReLU is chosen to be the best one as mentioned before. Colour, depth and normal maps were the best candidates with ReLU. As seen in the journal, both Sigmoid and Softmax have all of their predicted pick points clustered in the middle.

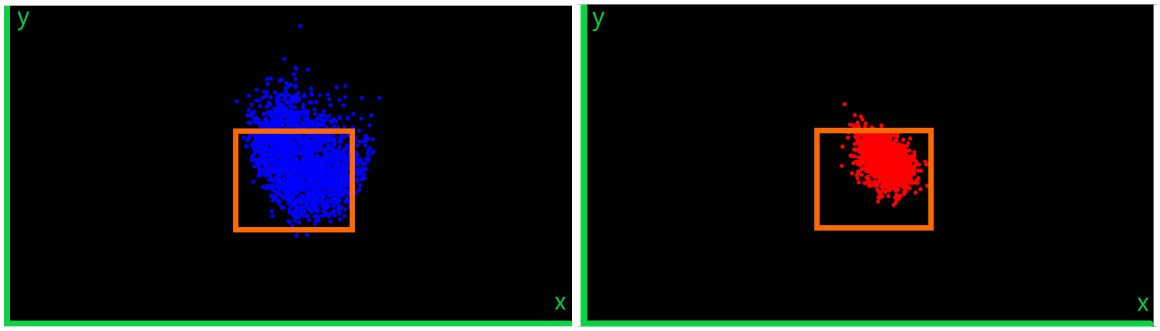
Loss Function

The mean absolute error (MAE) and the Huber loss are close to each other as in the mean squared error (MSE) except from normal maps.

– MAE and the Huber loss are the best loss functions for almost every type of image. However, MSE is the best one to use when the output of the network is a real number.[8] MSE is also mentioned in section 4.

The average distance is not the only parameter that was observed during the tests. To find a correlation between the ground truth and the predicted pick points, plots of the points are displayed next to each other in figure 10. The example shown below is from the test with a batch-size of 16 and colour as image type. The dimensions of the axes are:

$$0 \leq x \leq 848[\text{pixels}], \qquad 0 \leq y \leq 480[\text{pixels}]$$



(a) *Ground truth.* (b) *Predicted pick points.*

Figure 10: *10,000 pick points for the ground truth and for the model.*

The region of interest (ROI), as seen in the orange square on the figure above, points to the same area as the pick area. The ROI is used for the auto exposure time on the camera. The pick area is within the ROI, and this is the area the robot is allowed to pick within. This area is trapeze-shaped instead of square-shaped. The size of the pick area changes based on the load level of the bin. It is mostly in the left bottom of the ROI if the bin is at low level, and moves to the top right as the bin fills up. The pick area can actually move a bit out of the ROI as the camera's field of view gets narrower. Any pick point outside the pick area is discarded. The points outside the ROI in figure 10 are inside the pick area, which is why these points were *not* discarded. The predicted pick points in the image are clustered in the middle of the ROI, which means that the network has figured out that it is best to pick at that location.

Figure 11 shows an example of the pick area within the ROI, where it can be seen that the pick area is a bit outside the ROI.

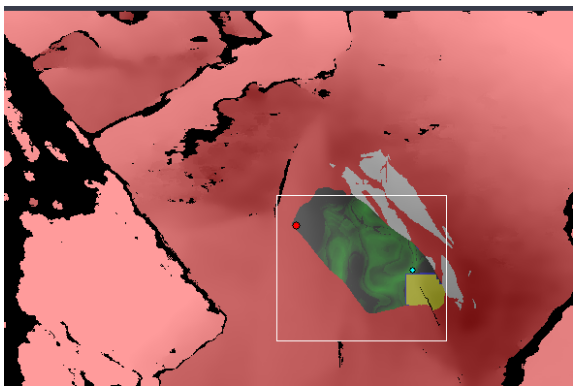


Figure 11: *The pick area marked as green-ish within the ROI marked with a white square.*

After each test, the loss curve was examined as well. Figure 12 show the loss curve for the model. Together with the loss curve, the learning rate has to be

configured. The learning rate controls the weights of the network with respect to the loss. The smaller the rate, the more minima will be found during training. However, it will also slow down the convergence. If the rate is too big, some minima will be missed, and could result in divergence. The learning rate should be in between those extremes.[9] The example below has a learning rate of 10^{-5}, which was chosen in section 4.

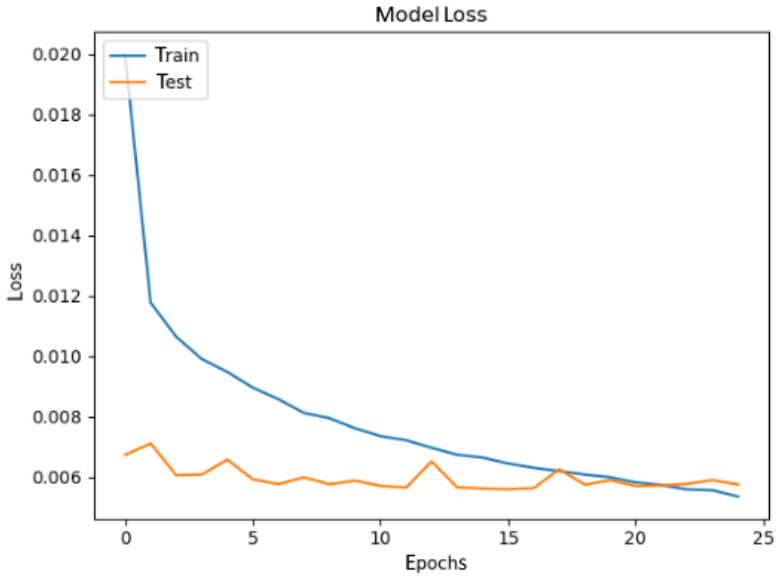


Figure 12: *The loss function of the model.*

5.3 Partial Conclusion

It is now possible to conclude which image type(s) and hyperparameters were the best ones compared to the average distance between the ground truth and the predicted pick points. These should be used for tests on the robot at Inwatec.

- Colour and depth are the best image types at 10,000 images.
- Depth and normal maps are the best image types at 25 epochs.
- Colour and depth are the best image types at a batch-size of 16.
- Colour, depth and normal maps are the best image types with ReLU as activation function.
- Colour, depth and colour+depth are the best image types with MSE as loss function.

Overall, the best image types with the best hyperparameters are:

- **Image types:** Colour and depth.
- **Hyperparameters:** 10,000 images, 25 epochs, batch-size of 16, ReLU as activation function, and MSE as loss function.

5.4 The Chosen Models

From the aforementioned conclusions, ten different models are chosen; five different models using colour images and five corresponding ones using depth images. They are all CNNs used for regression with the same configured hyperparameters. The main difference between them is the structure of the convolutional layers, pooling and filters. Listing 4 shows eight of the models. One is the base network *without* the layers marked with `// Added layer`; another a network *with* the layers marked with `// Added layer`[4]. The third is the base network with double filters except at the output layer, and a fourth with the "`// Added layer`"-layers and double filters[5]. The other four are the same, but for the depth images instead of colour.

Furthermore, two more models were constructed with additional convolutional layers and with a larger number of filters. This one is called VGG16[6]. All of these models had an average distance ranging from 55 to 65. To see if there really is a connection between the average distance and the performance of the models, two additional models[6] were constructed with 72 and 91 as average distance, respectively.

```
model.add(Conv2D(16, kernel_size=3, activation='relu', input_shape=input_shape))
model.add(Conv2D(16, kernel_size=3, activation='relu'))     // Added layer
model.add(Conv2D(16, kernel_size=3, activation='relu'))     // Added layer
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Conv2D(32, kernel_size=3, activation='relu'))     // Added layer
model.add(Conv2D(32, kernel_size=3, activation='relu'))     // Added layer
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(Conv2D(64, kernel_size=3, activation='relu'))     // Added layer
model.add(Conv2D(64, kernel_size=3, activation='relu'))     // Added layer
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Dropout(0.25))
model.add(Dense(128))
model.add(Dropout(0.25))
model.add(Dense(2, activation='linear'))
```

Listing 4: *Some chosen models.*

These 12 models can now be tested on the robot. However, the implementation of the models has to be configured first, which is described in the next section.

[4] Referred to as VGG1.
[5] Referred to as VGG2.
[6] Referred to as "bad" models.

6 Implementation

The models chosen in the previous section have to be tested on the robot at Inwatec. For this to be realised, the models have to be implemented using TensorFlow Serving. Afterwards, the models can be used with the separator-program, and the predicted pick points can be sent directly to the robot.

6.1 TensorFlow Serving

TensorFlow (TF) Serving is a serving system used to put models into production. The model is saved as a TensorFlow SavedModel. In TensorFlow 2, it is done as follows:

```
from tensorflow.keras.models import save_model
filepath = './saved_model'
save_model(model, filepath)
```

Listing 5: *SavedModel in Python.*

The structure of SavedModel is listed as below:

- saved_model
  - assets
  - variables
    - variables.data-00000-of-00002
    - variables.data-00001-of-00002
    - variables.index
  - saved_model.pb

TensorFlow Serving is used with Docker, and once this is installed, a Docker container can be created using the serving image. The Docker running TF Serving with the SavedModel is initialised through a self-made bash-script as seen in listing 6.

```
docker run --rm --init -v
/home/troelszink/Documents/BA/
Tensorflowserving/0002/saved_model:/ // Path to SavedModel
models/point_prediction/0001 \       // Target
-p 8501:8501 \                       // Port
-t tensorflow/serving \
--name tfserving_PP \                // Name of the serving image
--model_name=point_prediction \      // Name of the model
--model_base_path=/models/point_prediction
```

Listing 6: *Docker and TensorFlow Serving bash script.*

Client

The communication with TensorFlow Serving is configured within two separate classes. The first one called `HttpClient`, which handles the communication to the server, and the second called `Serving`, which handles the preprocessing of the images; the conversion of the image to a string of JSON-format; and sending this string to the `HttpClient`.

The neural network model requires an image as input. This image is an array of pixels with three channels. However, a depth image has only one channel, but as the model of the neural networks expects three channels, the value at each point in the depth image is copied to all three channels. The image is used within a function that makes a request to the container with the specified port number. Furthermore, it receives the response for the request, which is the predicted coordinate for the robot to pick. However, for this to work, the image has to be configured in valid JSON-format. A code snippet of the JSON-format is given below:

```
{\"instances\" : [[
[
    [0.000000,0.000000,0.000000],[0.000000,0.000000,0.000000],
    [0.000000,0.000000,0.000000],[0.000000,0.000000,0.000000],...
    [0.756863,0.784314,0.784314],[0.611765,0.607843,0.643137],
    [0.458824,0.482353,0.509804],[0.690196,0.713726,0.741176],...
]
]]}
```

Listing 7: *An example of the JSON-format.*

The code snippet shows that there are three channels per pixel in the image. Every channel is normalised from zero to one. When every pixel has been run through, the entire string, in JSON-format, is sent to the server. The predicted (x,y)-coordinate is located and saved from the output, as the output contains something that is not relevant. Both values are returned to the separator-program.

6.2 Configuration

The colour- or depth image is served using the function `serve(Mat image, string type)` called on an object of the `Serving`-class in the separator-program. The x- and y-values are then saved to a struct of training images. Afterwards, the two values are used to find the index of the pick point in the point-cloud; hereafter the robot knows where to go.

However, if the index (`maxidx` in the code) becomes below zero or above the highest

allowed index, then the `maxidx` is set equal to -1, which indicates that the robot should not be picking at that location. This was implemented, as it is possible that the model will predict a value that is not within the pick area. It also prevented a lot of issues with the program crashing every time it happened.

Another issue is that the robot will occasionally see itself when attempting a pick. To prevent this, a restriction of the z-axis of the robot was implemented. The program may only serve an image, if and only if, the robot is above a height equal to 1090mm. This way the robot does not see itself. A code snippet is shown below in listing 8.

```
trainingImage* tI = new trainingImage();
if (in->robotz > 1090) // So the robot does not see itself
{
    tI->pickID = coms->pickpointId;
    tI->color = colorMat->color;
    tI->depth = Mat(Size(848, 480), CV_8UC1);

    ...

    Serving S; // Sending a colour image to TF Serving
    vector<double> coord = S.serve(colorMat->color, "rgb");

    tI->x = coord[0]; // Saving the coordinate into the struct
    tI->y = coord[1];

    maxidx = tI->y * 848 + tI->x; // Used to find the point in the point-cloud

    if (maxidx >= 0 && maxidx < pixelNr) // Index within the ranges
    {
        int ximg = min(599, max(int(points[maxidx].y * 1000 + 300), 0));
        int yimg = min(599, max(int(points[maxidx].x * 1000 + 300), 0)
        int limitIdx = ximg * 600 + yimg;

        // Makes sure that the robot does not crash to the bottom of the bin
        if (points[maxidx].z < 0.002 || limitpt[limitIdx] != 255)
            maxidx = -1;
    }
    else
        maxidx = -1;
}
```

Listing 8: *Implementation in the separator-program.*

6.3 Flowchart

Figure 13 shows the flowchart of the entire implementation.

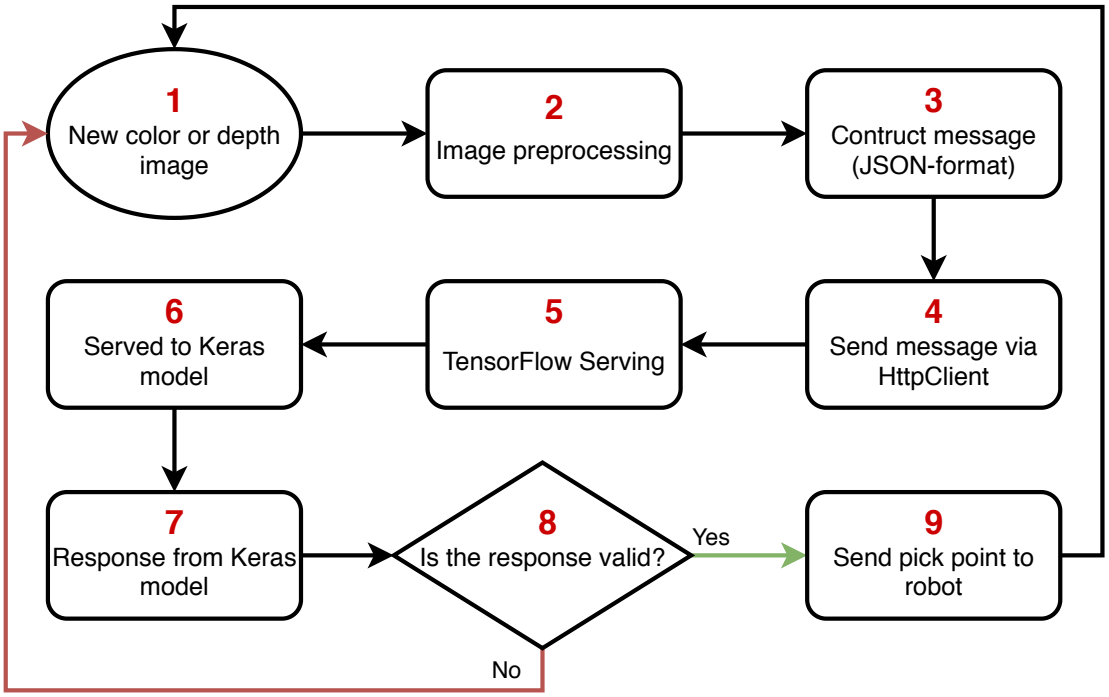


Figure 13: *Flowchart of the implementation.*

A detailed explanation of the flowchart is given below. Every process is referenced through the number given in figure 13.

1) First, a colour- or depth image in OpenCV `Mat` format is sent using the `serve(...)`-method along with the image type, as there is a difference in how the specified type should be handled in the method.

```
Serving S;
vector<double> coord = S.serve(colorMat->color, "rgb");
```

Listing 9: *Serving using a colour image.*

2) Inside the `serve(...)`-method, image preprocessing is done. The image is resized to a quarter of both the x- and y-value, resulting in 1/16 area of the original image. The model expects this size, as the neural network could not be trained with the original size due to lack of memory on the hardware used. Afterwards, the pixel values in the image are converted to float values and normalised.

```
vector<double> Serving::serve(Mat image, string type)
{
...
resize(image, image, Size(212, 120), INTER_AREA);
image.convertTo(image, CV_32F, 1.0/255.0, 0);
...
}
```

Listing 10: *Image preprocessing.*

3) The model expects an array of all the pixels in the image with three channels per pixel. A prefix and a suffix have to be added to the array in order to work. In this way, the format will be in JSON. The prefix and suffix are, respectively:

```
string pre = "{\"instances\" : [[";
string suf = "]]}";
```

Listing 11: *First- and last part of the JSON.*

The part in the middle will be added in a double for-loop (one for the y-value, and one for the x-value). Every pixel will be accessed, and three channels will be added to one pixel; the same channel copied three times in the depth type. The colour type is shown below:

```
Vec3f pixel = image.at<Vec3f>(y, x);
if (x < image.cols - 1)
    p = "[" + to_string(pixel[0]) + "," + to_string(pixel[1]) + ","
            + to_string(pixel[2]) + "],";
else
    p = "[" + to_string(pixel[0]) + "," + to_string(pixel[1]) + ","
            + to_string(pixel[2]) + "]";
```

Listing 12: *The middle part of the JSON.*

4) The string in JSON-format is appended to the `pre`-string and sent to the `HttpClient`-class using the `post(...)`-method. The response is saved in a string called `message`.

```
inwatec::HttpClient client;
string message = client.post("", pre);
```

Listing 13: *Sending a message using `post` from `HttpClient`.*

5) The connection to TensorFlow Serving is established as shown in listing 6. When the server runs, it is possible to send an image in JSON-format to it.

6) TensorFlow Serving uses the string in JSON-format according to a specific SavedModel file. In listing 6 the path to the SavedModel file is specified. This structure of this file is shown in the itemised list just above that listing.

7) The response from the model through TensorFlow Serving is saved in a string. This `message` has the form:

```
{    "predictions": [[x.xxxxxxxxx, y.yyyyyyyyy]    ]}
```

The x- and the y-coordinates are needed separately, and are then located and converted to a double. See the code snippet below:

```
inwatec::HttpClient client;
string message = client.post("", pre);

double coordx = stod(message.substr(22, 11)) * 848.0;
double coordy = stod(message.substr(35, 11)) * 480.0;
```

Listing 14: *Saving the pick point into an x- and y-value.*

8) The `coordx` and `coordy` from the response/`message` are checked to see, if they are valid or not. This check is done in the separator-program as seen in listing 8, where it is assured that the pick point is within the pick area using the `maxidx`.

9) Using the index (`maxidx`) the robot picks the corresponding point in the point cloud.

```
Serving S;
vector<double> coord = S.serve(colorMat->color, "rgb");

tI->x = coord[0];
tI->y = coord[1];

maxidx = tI->y * 848 + tI->x;
```

Listing 15: *Sending the pick point to the robot.*

6.4 Partial Conclusion

TensorFlow Serving has been set up successfully along with two extra classes added to the separator-program, that being the `HttpClient-` and `Serving`-classes. An extra chunk of code has been added to the separator-program inside an already existing class, which calls a function, `serve(Mat image, string type)`, within the `Serving`-class. A `Mat` image and the type of the image are given as arguments to the parameters of the function. The image is converted to JSON-format and sent to TensorFlow Serving, and a pick point is estimated.

7 Test

In this section, the chosen models are tested. Until now, the average distance between the ground truth and the predicted points has been the best way to measure the performance of the neural network. A lower average distance is expected to yield a better result. However, it cannot be guaranteed that model with the lowest average distance, predicts the best possible pick point. Therefore, the models have to be tested on the robot itself to examine the effects.

7.1 Purpose

The purpose of the tests is to examine the performance of the models chosen in section 5. Along with the chosen models, the acquisition-program used to acquire the ground truth, and the original separator-program are also tested. This is done to compare the performance between these programs and the models.

The success rate is used to compare the performance of these programs and models. The success rate is a metric (measured in percentage) determining the number of successful pick attempts. This rate is defined by Inwatec as pick attempts with the scores of 1, 2 and 3. Furthermore, the speed of the robot is measured in numbers of garments per hour, that is, how many garments are delivered per hour. The speed is not essential for this project. However, the speed can be used as a secondary metric to evaluate the performance.

7.2 Results

Table 3 shows all the models along with the original separator-program and acquisition-program. All of the tests are done with the same type and amount of laundry inside the bin. All are tested with 500 pick attempts to ensure convergence.

Table 3: *The original and acquisition program together with the chosen models tested on the robot.*

	Type	Average Distance	Garments per Hour	Success Rate
Original	–	–	1650	98.0%
Random	–	–	1275	71.0%
Base1	Colour	56	1250	82.4%
Base2	Colour	58	1275	85.2%
VGG1	Colour	55	1300	83.4%
VGG2	Colour	56	1300	83.8%
VGG16	Colour	60	1275	85.4%
Base1	Depth	65	1275	82.6%
Base2	Depth	55	1325	88.0%
VGG1	Depth	59	1200	82.4%
VGG2	Depth	59	1325	81.6%
VGG16	Depth	59	1400	85.6%
Bad Model	Colour	91	500	84.0%
Bad Model	Depth	72	1275	82.0%

The original separator-program has a success rate of 98.0%. The acquisition-program has 71%, which is relatively high while choosing random pick points. However, all of the models tested are above the acquisition-program, but below the original one. All of the models are ranging from 81.6% to 88.0%.

The four models that performed the best are marked with green in the table. These four are `Base2` and `VGG16` in both image types. The bad model with colour images was only tested on 200 images and had a slow speed. This model predicted a large number of pick points outside the pick area, which resulted in a rejection of those points. This in turn meant that the robot required manual intervention in order to continue with another image.

After all of these models had been tested, the best one, `Base2` for depth images with a success rate of 88%, is further developed to improve the success rate even more. The goal is 90%.

Alongside the two configured versions of the `Base2`-network, two ResNet models are tested as well; one with a frozen backbone (from transfer learning, TL). These four models are shown in table 4.

Table 4: *The extra four models.*

	Type	Average Distance	Garments per Hour	Success Rate
Original	–	–	1650	96.8/91.4% (5.4)
Random	–	–	1275	71.0/69.0% (2.0)
Base2_v2	Depth	55	1450	80.8/77.8% (3.0)
Base2_v3	Depth	54	1500	83.6/80.8% (2.8)
ResNet TL	Colour	61	1250	83.2/77.6% (5.6)
ResNet Own	Colour	76	750	82.3/77.4% (4.9)

In these four tests it was decided that the success rate of only the score of 3 would be examined as well as for the score 1, 2 and 3 combined. The success rate of the score equal to 3 is shown in the right part of the column containing the success rate. This was done with the original and random program as well. The success rate of the score equal to 3 is quite close to the original rate. However, the difference (the values in parenthesis) between these two is different in each model. Nevertheless, the difference in the original program is larger than in the configurations of the `Base2`-network. Notice that the success rate of the original program is 96.8% instead of 98% from table 3, as a small deviation can occur.

The average distance in proportion to the success rate for all models is plotted in figure 14. It shows no linear nor polynomial correlation. The closest fit is a 9th degree polynomial that results in $R^2 \approx 0.26$. However, from the figure it seems that a linear line could be fitted from a distance of 61 and upwards. There is no overall correlation, but there could be one, when the distance is high enough. The higher the average distance is, the higher is the risk for the pick points to be outside the pick area, which will in turn lower the success rate and the delivery speed.

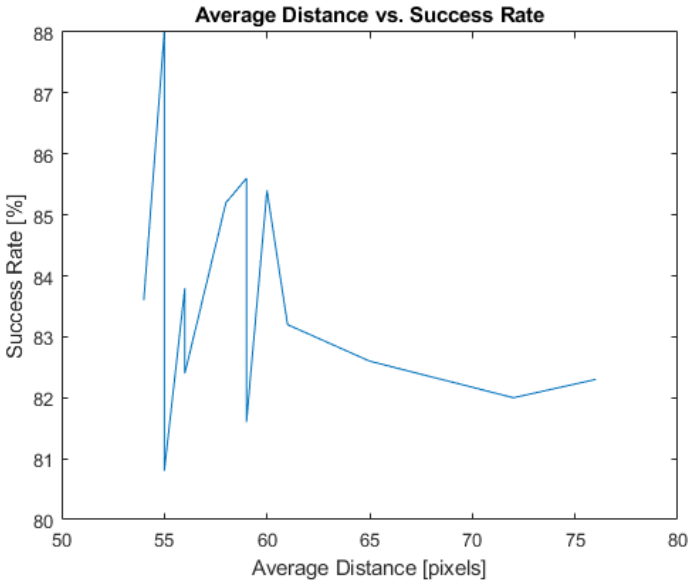


Figure 14: *Average distance vs. success rate with every model except* `Bad Model` *in colour.*

Figure 15 shows the plot of all models including ground truth, but excluding the two "bad" models. There are 25 points for every model, each representing an image. Every image is chosen uniformly from lowest to highest x-value from ground truth. Each of the 25 images has been served with TensorFlow Serving for each model to compare the outputs. The ranges of the axes are limited to $250 \leq x \leq 650$ and $100 \leq y \leq 400$. There are a few points outside this area, but these are interpreted as outliers. The orange square is the region of interest, as it is not possible to show the exact pick area, as it changes for each image.

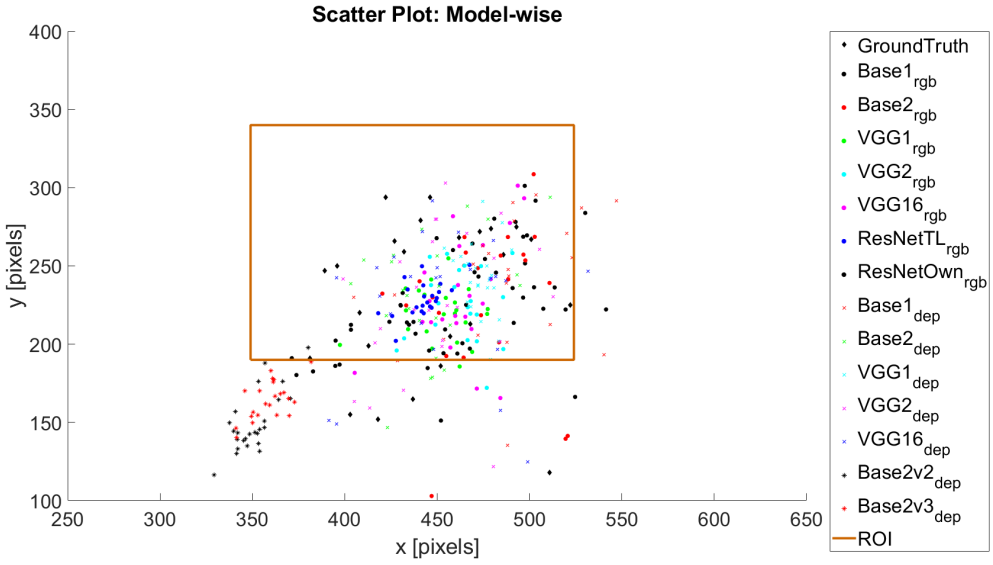


Figure 15: *Model-wise plot of 25 test images for each model.*

As there is no significant direct correlation between the average distance and the success rate, it is difficult to see why the models with high success rate are doing better than the other ones.

When looking at figure 15, it is possible to see that the models with the highest success rate tend to be within the ROI. Figure 16 shows how many models have a certain number of points inside the ROI.

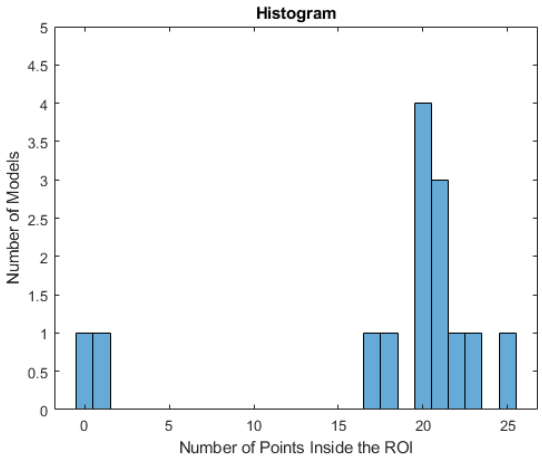
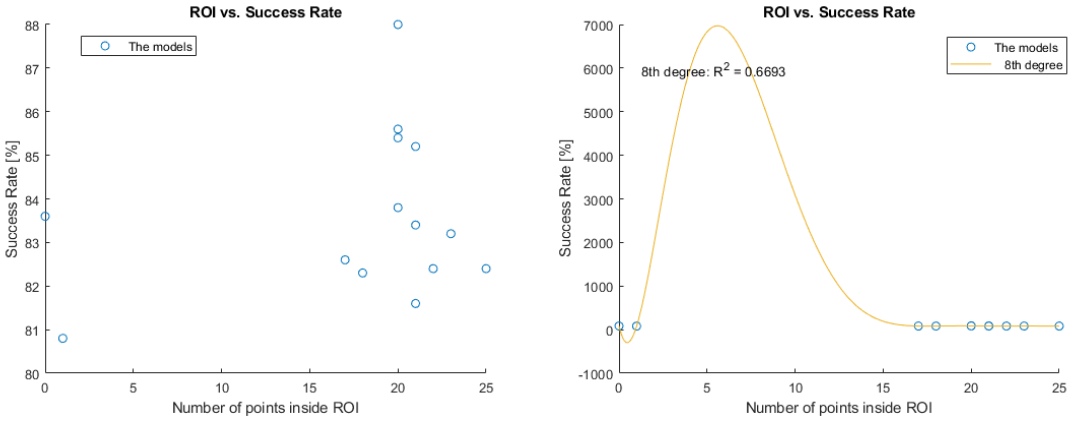


Figure 16: *Histogram of the models showing the number of points inside the ROI.*

Figure 17 shows the number of points inside the ROI compared to the success rate.



(a) *Pick points inside the ROI.* (b) *8th degree polynomial fitted to the models.*

Figure 17: *Number of points inside the ROI compared to the success rate of the models and with an 8th degree polynomial fitted to the models.*

All of the models have 17-25 points inside the ROI except for two models; one with one point with the worst success rate at 80.8%. The best model has 20 points inside, which means that there is no direct correlation. An 8th degree polynomial can be fitted to the data with $R^2 = 0.67$, which is a better correlation than before with the average distance.

Figure 18 shows the four models that have 20 points inside the ROI (see figure 16). One of these is the best network, `Base2_dep`, with a success rate of 88.0%.

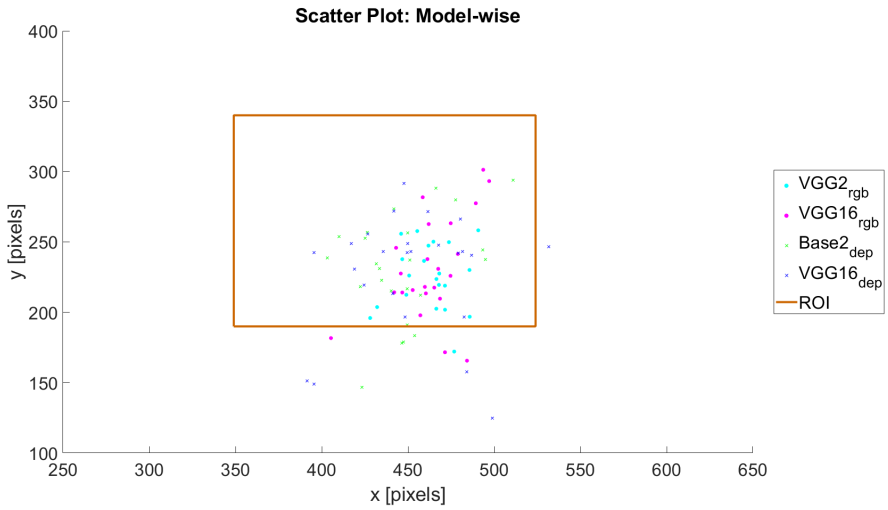


Figure 18: *Four models with the same amount of points inside the ROI.*

To investigate the correlation between the success rate and the dispersion of the points within the ROI (with the same number of points inside), a method to calculate the dispersion is developed. The dispersion is calculating by taking every point and calculate the distance from this point to every other point in the same model. This is done for all four models. Figure 19 shows this compared to their corresponding success rates. This correlates with $R^2 = 0.75$, which shows that there is a likely correlation between the dispersion of points within the ROI.

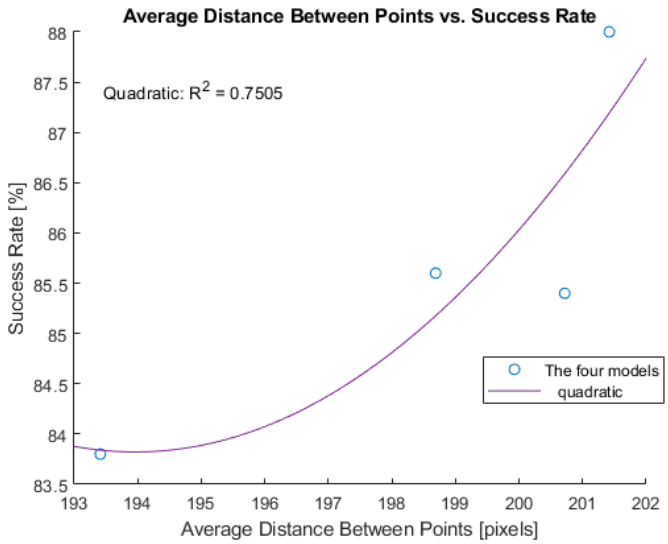
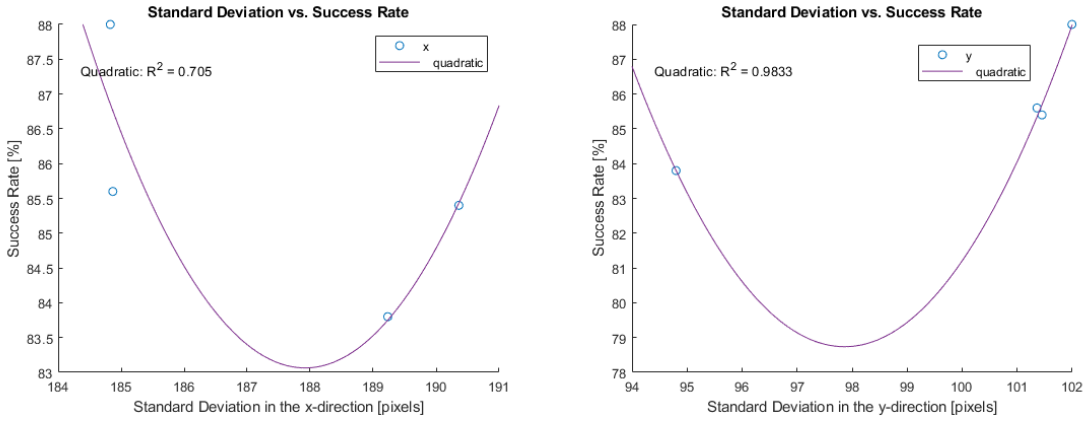


Figure 19: *The four models with the same number of points inside the ROI compared with the average dispersions and success rates.*

The standard deviations of the four models are considered as well. The standard deviation is determined for all of the 20 points inside the ROI for each of the four models. It is done for both x and y to see if there is a correlation in the dispersion of x- and y-values. Figure 20 shows the standard deviations compared to the success rates.



(a) *Standard deviation in the x-direction.* (b) *Standard deviation in the y-direction.*

Figure 20: *Standard deviation of the x-values and y-values for each of the four models.*

The standard deviations in the x- and y-direction are fitted with a quadratic polynomial with $R^2 = 0.71$ and $R^2 = 0.98$, respectively. The standard deviation is likely a better tool for estimating the dispersion of the points.

7.3 Partial Conclusion

The success rate of the acquisition-program resulted in 71%, which is high in proportion to its randomness. Thus beating this would mean that it is possible for the models to learn a correlation between the images and good pick points. All models resulted in a higher success rate, ranging from 80.8-88.0%, than the acquisition-program, but lower than the original separator-program. However, when excluding partial successes the original separator-program only achieved a success rate of ≈92%. This is relatively close to the best model when including partial successes at 88%. The best model was not tested without partial successes.

It is not possible to see a direct correlation between the well-performing models and the points predicted. However, there is a likely correlation between the points inside the ROI and the success rate, and a less likely correlation between a model's average distance and success rate. It also seems that the well-performing models have a larger dispersion of the points inside the ROI instead of points clustered together.

8 Discussion

In this section the choices made during design, training and testing will be discussed. Alternative solutions will be explored along with the consequences thereof.

To simplify the input of the network, the index of the pick point could have been used instead of the (x,y)-coordinate; this would result in only one label instead of two for each coordinate. The index calculation is as follows: index $= x \cdot 848 + y$. This would allow for a single output from the network and would also require one less step in the implementation with the separator-program.

When training the network, the Keras API could have been configured to return metrics such as accuracy and loss that can be used to evaluate the network. Normally the accuracy can be used to evaluate the performance of the network, and therefore a higher accuracy should yield a higher success rate. However, when using a CNN for regression, the accuracy of the network only indicates how many times the ground truth were correctly predicted. With a regression network multiple points in the image could have the same properties of the ground truth. This is due to the fact that the network looks at the correlation between each point in the image and the ground truth. Thus only the loss metric can be used when evaluating the network.

To solve the problem regarding models with a high average distance, the input to the network could have been modified. The input currently consists of a full-scale image from the camera, but this image could have been cropped before training to only contain the region of interest. This could eliminate predictions outside the ROI and improve the speed of all networks. Changing the size of the input image would also allow for a better resolution, as the current input images are scaled down to 1/16 area of their original size to conserve memory. Thus this could be omitted and more data in the image could be preserved and help the network obtain better results.

During the training of the network, the average distance between the ground truth and the predicted values are calculated. This distance was found to be roughly the same for most of the models, ranging from 54-65, most being below 60. Models with an average distance that deviated greatly from this were also tested. Following these tests, it can be seen that the models with the same average distance do not achieve the same performance. With a low average distance, the predicted pick points are more likely to be inside of the ROI and therefore be viable pick points. A higher average distance would result in more predictions outside the ROI, which could result in a slower pick rate, as some predictions

might not be viable pick points and must be discarded. This indicates that the average distance cannot be used as a metric to evaluate the performance. Instead, the amount of predictions inside the ROI could be used as a metric, since tests have shown that models with more predictions, along with greater dispersion, inside the ROI achieved better performance.

One problem that occurred during testing of the models on the robot was the implementation of the model into the separator-program. It posed challenges, as the separator-program is quite complex, and there are still some problems that arise when running the program with the model. The largest problem occurs when the program sends an image containing a piece of garment hanging from the grippers of the robot or from the delivery conveyor belt, as this can cause the network to predict a point in the image containing a moving piece of garment. This may result in a failed pick attempt due to movement of the garments.

Furthermore, to better compare the performance of the models with the performance of the original separator-program, specific scenarios could have been devised. These scenarios would test the two programs on different aspects, as the models might perform worse on plain garments such as towels, but it might perform better on heavy duty work cloths or other garments. The goal would be that the models offer a more robust solution than the original separator-program.

Lastly, although the models tested on the robot worked quite well considering their size, they could have been expanded and bigger models could have been trained. The most limiting factor is the memory usage; however, this could have been greatly reduced if the images had been cropped to the ROI as previously discussed. This would allow for larger models to be implemented, and those models might be able to extract more complex features that could help improve prediction and increase the success rate. However, this would result in a longer computation time, but this could be alleviated running the model on a GPU.

9 Conclusion

A convolutional neural network has been designed to perform regression; this network accepts colour and depth images as input, and outputs a pick point consisting of an x- and y-value. Each image is handled by a specific model through TensorFlow Serving. The model sends the pick point back through TF Serving. This pick point is used within the separator-program to find the corresponding point in the point cloud, which in turn moves the robot to that location.
Below are the four topics discussed in the Problem Description concluded upon.

Data Acquisition and Preprocessing

- The type of data needed was both colour and depth images. Both types were examined to see if the neural network would see a clear pattern in the depths instead of the colours. Both types were examined in addition to normal maps and a combination of the two. The two models with the highest success rates used depth images, but it does not seems to be consistent that depth images were better than colour.

- All tests were done using 10,000 images. However, in the tests of the hyperparameters, five different number of images were examined, and the best model fit was at 5000-10,000 images with colour and depth. This was concluded based on the average distance.

- Because of hardware limitations, preprocessing was needed. The images were rescaled to reduce memory usage, which decreased the trainable parameters to $212 \times 120 \times 3 = 76,320$. Furthermore, normalisation was implemented to ensure that the data was within the same scale.

Design of Network

- A convolutional neural network capable of performing regression has been designed to use an image as input and output a pick point consisting of a 2D-coordinate. 2D CNN-layers were added to the network along with Maxpooling- and Dense-layers. The output layer has two neurons to ensure two values at the output.

Training and Test of Network

- The neural networks had to be trained in order to achieve the highest possible performance. This was done by tuning the hyperparameters to see how they affected the performance. The goal was to lower the average distance between the ground truth and the corresponding predicted pick points for every image. This was tested on the robot at Inwatec; however, the tests showed no apparent correlation between

average distance and performance. The network was also expanded with more convolutional layers, more pooling layers and a higher number of filters; however, better results were not obtained. The number of points inside the ROI and the distance between every point in the same model were examined, and a correlation was found.

- The tuning of hyperparameters involved number of images, number of epochs, batch-size, activation functions, and loss functions. The chosen values for each type of hyperparameters were used in the tests on the robot. This was done to have a foundation to work on.

- The performance of the neural networks was quantised by measuring the success rate including the scores of 1, 2, and 3. It was also tested with only the score of 3 on a few models. The average number of garments per hour was also examined; it would be used as a metric if two models had the same success rate.

Analysis of Model Performance

- The best model tested on the robot was the `Base2`-network with depth images that reached a success rate of 88.0%. This is 17% percentage points above the acquisition-program, but 10% percentage points below the original separator-program. This shows that the model is able to learn the correlation between the ground truth values and the image features.

10 Perspectives

Further improvements to the project could have been made if time had been abundant. Such improvements will be discussed and described in this section.

The first improvement that could have been made is changing the network architecture from a regression neural network to an encoder-decoder network. Doing this would alleviate one of the main problems that this regression neural network suffers from, which is stagnation in the change in the input image. The encoder-decoder network would return a heat map as an output instead of an (x,y)-coordinate. This would allow for more careful selection of pick points within the heat map. The encoder-decoder network works by first using a downsampling network structure to encode the information in the image into the pixels, and then using an upsampling network structure. This upsampling network will encode the spacial information in the heat map, and this will then show the location of a good pick point with a certain likelihood. A visualisation of a heat map can be seen in figure 21.

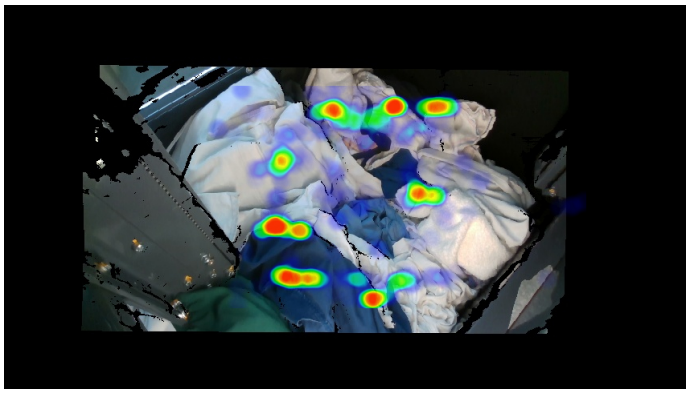


Figure 21: *A visualisation of a heat map on top of an image from the robot.*

As can be seen in the figure, a heat map would allow for more options for the pick point, as each peak (red area) in the heat map would be a good location of a pick point, thus allowing for a broader selection and avoiding stagnation, when using a single prediction.

Another change that could have been made to the network is the data trained upon. Instead of predicting 2D-points in an image, it could have been expanded into predicting 3D-points for the robot to pick. This would be done by defining a 3D bounding box that the network is allowed to predict within, and then feeding it the depth images. This would alleviate the problems with hanging garments, as they will either be out of the 3D bounding box or be included in the training data, and the network will be taught not to predict points at those locations.

Another way of avoiding the hanging garments could be to teach the network to track the state of each piece of garment. This would allow it to keep multiple pick points ready for each piece it locates, and then choose the best of these candidates. This would also solve the problem with the hanging garments that has been discussed earlier, as this would allow the network to know if a garment is moving or not. This would, however, require a more complicated network and more computer resources to run. Furthermore, this would also require advanced training data.

11 References

[1] Adam. (s.d.). Keras Documentation. Localised the 22.05 2020 at `https://keras.io/api/optimizers/adam/`

[2] Brownlee, J. (s.d.). Difference Between a Batch and an Epoch in a Neural Network. Machine Learning Mastery. Localised on 14.05 2020 at `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/`

[3] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. Localised at `https://arxiv.org/abs/1412.6980`

[4] Inwatec. THOR – Robot Separator. Localised at `https://inwatec.dk/product/robot-separator/`

[5] Normal mapping. (s.d.). Wikipedia. Localised on 14.05 2020 at `https://en.wikipedia.org/wiki/Normal_mapping`

[6] Simonyan, K. Zisserman, A. (2015, April 10th). Very Deep Convolutional Networks for Large-Scale Image Recognition `https://arxiv.org/pdf/1409.1556.pdf`. University of Oxford.

[7] The Functional API. (s.d.). Keras Documentation. Localised on 26.05 2020 at `https://keras.io/guides/functional_api/`

[8] Verma, S. (s.d.). Understanding different Loss Functions for Neural Networks. Towards Data Science. Localised on 16.05 2020 at`https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718`

[9] Zulkifli, H. (s.d.). Understanding Learning Rates and How It Improves Performance in Deep Learning. Towards Data Science. Localised on 16.05 2020 at `https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10`

12 Appendix

Overview of the Electronic Appendix

This is an overview of the electronic appendix, which is submitted together with the report. The full separator-program cannot be shown due to an agreement with Inwatec. Only the modified part is shown in the Code-folder, together with the deep learning code, and the TensorFlow Serving code.

- Code
- Figures
- Bachelor Contract.pdf
- Collaboration Agreement - Jacob.pdf
- Collaboration Agreement - Troels.pdf
- Journal for Training.pdf
- Time Table.pdf