Can we predict grasp forces from photos?

Können wir Greifkräfte mithilfe von Fotos vorhersagen?
Bachelor-Thesis von Nick Heppert aus Heppenheim
November 2017

TECHNISCHE UNIVERSITÄT DARMSTADT

Can we predict grasp forces from photos?
Können wir Greifkräfte mithilfe von Fotos vorhersagen?

Vorgelegte Bachelor-Thesis von Nick Heppert aus Heppenheim

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Ing. Michael Goesele
3. Gutachten: MSc. Filipe Veiga

Tag der Einreichung:

For everybody,
who might be interested

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. lol Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 15. November 2017

(Nick Heppert)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, November 15, 2017

(Nick Heppert)

Abstract

Grasping objects with the right amount of force, neither to soft or to hard, is still a problem for robots. This thesis aims to estimate the necessary grasp force from a single photo. Therefore the force estimation is formally split into estimating the friction coefficient and the mass of the object. To predict the friction coefficient Convolution Neural Networks were used, with moderate results, because a strong correlation could not be learned. By taking new photos in a controlled environment the prediction could be improved. In order to deal with uncertainty of the predicted friction coefficient an extension is proposed by predicting a discrete probability distribution over all friction coefficients. The mass of the object is predicted by making use of a image segmentation mask and Gaussian Processes. To retrieve the final grasp force both predictions are multiplied. Experiments on the real robot showed that the quality of the estimation is mixed. Three out of eight experiments had a success rate below 50%.

Zusammenfassung

Das Hochheben von Objekte mit der richtigen Kraft, weder zu sanft oder zu fest, ist nach wie vor ein Problem fuer Roboter. Diese Arbeit zielt darauf ab, die notwendige Greifkraft nur anhand eines Fotos vorherzusagen. Hierfuer wird die Kraft zu erst in den Reibungskoeffizienten und die Masse aufgeteilt. Der Reibungskoeffizient wird mithilfe eines neuronalen Netzes vorhergesagt. Die Ergebnisse ueberzeugen nicht vollstaendig, da keine starke Korrelation gelernt werden konnte. Die Ergebnisse konnten jedoch mit neuen Fotos, welche in einer kontrollierten Umgebung aufgenommen wurden, verbessert werden. Um Unsicherheiten in der Vorhersage des Reibungskoeffizienten zu kompensieren, wird eine Erweiterung vorgeschlagen, indem eine diskrete Wahrscheinlichkeitsverteilung ueber alle Reibungskoeffizienten vorhergesagt wird. Die Masse des Objekts wird mit Hilfe einer Bildsegmentierungsmaske und "Gaussian Processes"vorhergesagt. Um die endgueltige Greifkraft zu erhalten, werden beide einzelnen Vorhersagen multipliziert. Experimente auf dem echten Roboter zeigten, dass die Qualitaet gemischt ist. In drei von acht Experimenten war die Erfolgsrate unter 50%.

Acknowledgments

I would like to thank Filipe Veiga for technical and mental support – Muito obrigada pela sua ajuda!
Furthermore I would like to thank Max von Buelow for helping me taking photos of the objects.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
API	Application Programming Interface
CE	Cross Entropy
CFM	Coulomb Friction Model
CNN	Convolution Neural Network
CV	Cross Validation
FCS	Friction Coefficient Scalar
GP	Gaussian Process
MSE	Mean Squared Error
NN	Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
ROS	Robot Operating System
YCB Object set	Yale–CMU–Berkeley Object Set

List of Symbols

Notation	Description
$\boldsymbol{\omega}$	Vector of **angular** velocities
B	Amount of **bins** for segmenting along a dimension
Im	**Image** of an object

v	Vector of **linear** velocities
$\boldsymbol{\theta}$	Hyper **parameters** for a kernel
Pa	Small **patch** of quadratic size
β	**Precision** of a gauss distribution
P_{DC}	Absolute fluid **pressure** measured by the tactile sensors
$\mathbf{I}_a$	**Unit matrix** with dimension $\mathscr{R}^{a\times a}$

List of Operators

Notation	**Description**	**Operator**
σ	Activation function of a neural network	$\sigma(\bullet)$
$\mathcal{N}$	Gauss distribution	$\mathcal{N}(\bullet, \bullet)$
$\mathcal{N}$	Probability of a sample from a Gauss distribution	$\mathcal{N}(\bullet \mid \bullet, \bullet)$
k	Kernel function	$k(\bullet,\bullet)$
log	Logarithm	$\log(\bullet)$

1 Introduction

Robots are becoming more and more important in our daily lives. To assist humans, robots should be able to perform dexterous manipulations tasks with their hands, but they still struggle to complete tasks considered easy for humans, e.g., lifting objects. Such tasks can be split into three main parts: Observing the scene, planning the movements and executing the task. When this rough scheme is specified for lifting objects, following procedure can be seen as state-of-the-art.
Modern object detection approaches can detect objects at a rate of 5Hz with a success rate of at least 70% [1]. However, when lifting an object with a robotics hand, detecting it in the scene is only the first step. After a successful detection the placing of the fingers onto the object and the lifting motion must be planned [2]. As a last step the robotics hand needs to apply sufficient forces on the object to successfully lift it up.
Yet, when executing this task, the last step is still a crucial problem for robots because they are not equipped with enough sensors to recover useful information when grasping an object. The necessary information would include data which could be used to gain the tactile feedback of the grasp.
To overcome this problem, tactile fingertip sensors have been developed [3]. With this new form of finger tips, the force, which is applied to the object, can be measured. It is even possible to predict or detect occurring slip, as shown in [4] and [5]. Without such sensors the robot has to rely, e.g., solely on his vision or torque sensors at the joints, to recognize if he has to increase the force to prevent a drop of the object. Alternatively the robot can assume just a fixed joint position configuration for its fingers without reacting to the consequences for the object. [4] and [5] developed controllers on top of their respective slip prediction or detection to control the force onto the object. With the help of this additional controller they successfully managed to lift different objects.

1.1 Motivation

In this thesis an estimation of the necessary force to grasp an object is proposed. With the help of a tactile sensor it is possible to measure the applied force onto an object. When the estimated force is reached across all tactile sensors on the hand, it is possible to successfully execute the last step as in the previous section described, lifting the object.
Briefly, the force is estimated by predicting two different variables, the friction coefficient and the mass of the object. The resulting product of the predictions multiplied by the gravity is proportional to the desired force. The objective of this thesis is to develop two predictive models, one that estimates the friction coefficient and one that predicts the mass of the object.

Is it really necessary to estimate the force in advance when such controllers exist to control the force by detecting slip?
When lifting an object, two possible extreme cases can occur. On the one side a fragile object, e.g. a banana or a plastic cup, might be damaged as applying too high forces could deform the object. On the other side, when applying too little force, the object might slip quickly through the hand of the robot. Such a problem might occur when you hand a heavy object, like a brick, to the robotic hand, which assumes as little force as possible to apply.

Why predicting from photos and not directly providing the necessary force?
In scenarios where no operator can provide a force estimation it is necessary to predict it. One of the simplest task for a robot is taking a photo of the scene, which is the easiest and fastest way to obtain a digital representation of the desired object. Hence, a prediction based on a photo or multiple photos can be done by the robot itself if the operator can not provide any insight. To generalize best, a prediction solely based on a single photo is desired. Nonetheless, multiple photos should increase the prediction quality.

1.2 Related Work

As already mentioned, [4] and [5] propose a method to predict or detect slip with the same BioTac fingertip sensors [3]. [6] and [7] suggest similar methods but with their own tactile sensors respectively. Despite the different slip prediction or detection, in general all grasp force controllers work the same: Depending on the slip state, the force onto the object is increased and the grasp is stabilized. The crucial point, where this thesis is going to assist these controllers, is in the beginning. At this point the thesis aims to predict the forces as accurately as possible so that the controllers do not need to adjust the grasp force. Existing controllers start with different forces applied to the object as outlined in the following.

[4] and [7] only start with the object in contact, the minimum force possible. As a result, the hand only touches the object. From that point on the force is subsequently increased while the object is assumed to be slipping.
In contrast to [4] and [7], [5] also start with a fixed value across all objects, this guess is based on known materials and their interaction with the sensors skin [8]. [5] even state "Since the initial friction coefficient is not estimated accurately, slip may occur during the lifting", what shows that there is room for improvement at the early stage of grasping.
[6] use a more sophisticated way to determine the initial force. In a preprocessing step, the "Tactile Exploration Phase", they slowly slide the finger over the surface of the object. Based on the results of the sensor, the friction coefficient is estimated. A similar approach was chosen by [9]. They propose a new finger tip sensor, which estimates the friction coefficient directly without sliding over the object, only by touching it. Yet, one thing that all of these methods have in common is the usage of tactile feedback in advance. Estimating the friction coefficient solely from photos, has not been deeply explored previously.
Regarding the second prediction, the mass estimation from photos, prior work is also sparse.
[10] predict the weight of cows by extracting various information, including the wither and hip height as well as the hip width and the body length. With the help of such properties they showed, it is possible to predict the weight of cows from photos.
A more image processing oriented approach is used by [11], where the weight of pigs is predicted. After taking only one image of the animal from the top, the image is segmented and the pig is masked out. Based on this segmentation mask properties like the area covered and the body length are checked for correlation with the weight. With the found correlation the weight for new pigs can be predicted. [12] successfully use a similar approach.
[13] suggest a different approach by using the image segmentation mask, but also detecting the edges of the object. With these detected edges they determine the height of the object and the area covered. The ascertained values help estimating the volume of the object and so the weight can be calculated.
Nevertheless, all described approaches have in common that the density is assumed to be known or fixed. [13] determine the density from a predefined table of densities of known objects. This approach only works for known objects, as unseen objects are not listed in the table. For the prediction of the weight of animals, the density is expected to be always equal for each animal type.

1.3 Outline

In Chapter 2 the foundations, which are necessary to understand the thesis, are presented. First, the background about grasping an object (Section 2.1) and second, the basic mathematics necessary for the model definitions (Section 2.2, 2.3) are explained. With this knowledge, the problem is formally defined in Chapter 3. The split between the different models (Section 3.1, 3.2 and 3.3) is explained and formulated. The used objects (Section 4.1), the robotic setup (Section 4.2), and the object survey (Section 4.3) as well as the attempt of solving the previously defined problem (Section 4.4) is described in Chapter 4. In Chapter 5, first, the results of the different models (Section 5.1 and Section 5.2) are discussed. Second, a theoretical evaluation in Section 5.3 was made. Third, to improve the results new improved photos of the object were taken (Section 5.4). Fourth, with the help of the new photos multiple final predictions (Section 5.5) were made and evaluated on the robotic setup to close the loop. Last, an overview over next possible steps is given in Chapter 6.

2 Foundations/Background

By analyzing the question "Can we predict grasp forces from photos" the desired task can be split into two parts. First, on the tactile side, one has to understand the basics of grasping an object to predict the forces. Second, one has to understand how to make a prediction based on photos.
In the following the required background for these two parts is given.

2.1 Grasping

Grasping of an object can be accomplished by several methods. For simplicity of explanation, two possible ways are presented in the succeeding.
The robot could either make use of the geometric representation of the object and perform a form closure on the object [14]. Or the robot could use the friction between the object and its robotic fingers by applying enough force to overcome gravity or any other external applied forces and moments, performing a force closure [15]. A comparison between the two types is illustrated in Figure 2.1, while a brief explanation of both is given in the following sections.

2.1.1 Form Closure

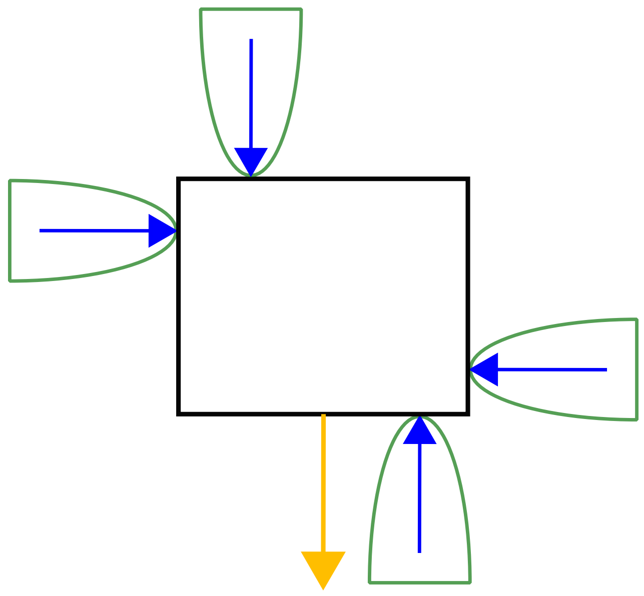
The form closure only relies on the placement of the contacts on the object. For each contact, a constraint is introduced, which is unilateral and frictionless. Therefore, the force is always normal to the surface at the contact. Because the constraints are unilateral and frictionless, for each contact a contact force on the object is given.
When in three-dimensional space the possible infinitesimal movements of the rigid object are defined by the three Cartesian linear velocities captured in v as well as by three Cartesian angular velocities in $\boldsymbol{\omega}$, the goal for a successful grasp in form closure is to find a set of contact forces that compensate external forces and moments including gravity. When either multiple forces or moments are applied on the object, a separate sum over all forces and moments can be made and seen as a single force and moment.
[14] showed that for a grasp in a two-dimensional space four contact forces are required to fully compensate the external forces and moments for any shape. An example of a possible object and a set of contact points in the two-dimensional space is shown in Figure 2.1a. [16] extended this generalization for the three-dimensional space, discovering that seven contact forces are needed to compensate all possible velocities.

2.1.2 Force Closure

In order to understand force closure, first, a very basic concept of friction is presented, the Coulomb Friction Model (CFM). Briefly explained, when a normal force $\vec{F}_\mathrm{n}$ with a magnitude of $f_\mathrm{n} > 0$ is applied, the resulting tangential force $\vec{F}_\mathrm{t}$ has a magnitude of

$$f_\mathrm{t} \leq \mu f_\mathrm{n} \tag{2.1}$$

but not less then zero. The direction of the force is always against a possible movement. The variable μ, the friction coefficient, is empirically determined between the two contacting materials [17].
With the help of the friction coefficient a friction cone can be constructed at the point of contact. The cone, with an opening angle equal to $\mu/2$, is the area in which the direction of the force contact must be located. If the contact force direction is within this cone, it is guaranteed that the finger does not move on the surface of the object. As a result, the tangential contact force is applied onto the object and can compensate external forces and moments. However the object might still slip if the tangential force is too small because the compensation capabilities are limited by Equation 2.1.
An extension to the CFM, which can be made, is the soft-finger contact. Using this contact model an additional moment around the normal axis is introduced besides the occurring tangential force. Like the tangential force the moment can be calculated by a coefficient, the torsional friction coefficient [18].
For the simplicity of the model, only the CFM is used and it is assumed that the applied force is always normal to the object. A force closure grasp is now defined by its normal contact forces and their resulting tangential contact forces. In comparison to the form closure grasp, here, a contact results in two contact forces instead of one single contact force. Once again, the grasp can be categorized as successful if all external applied forces and moments can be compensated with the help of the contact forces.


(a) Form closure in a two-dimensional space using four contacts (green) and the resulting contact forces onto the object (blue)

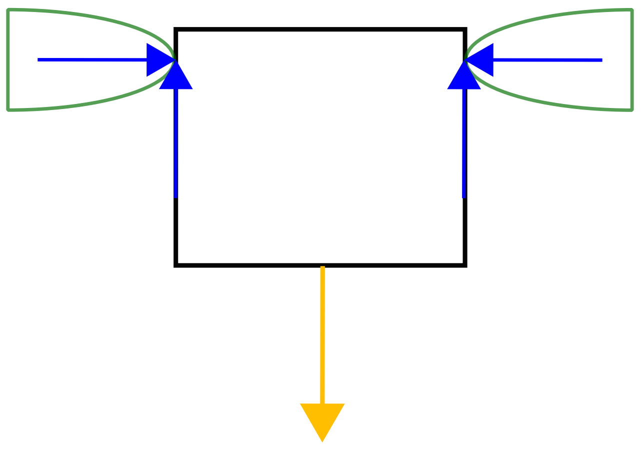

(b) Force closure in a two-dimensional space using two contacts (green): When using the Coulomb Friction Model the two contacts could be modeled like the form closure (blue).

Figure 2.1.: Form closure and force closure in comparison: For both grasps the four contact force constraint shown in [14] is satisfied and the resulting forces onto the object can compensate the external force (yellow). *Please note that the length of the force vectors do not indicate the magnitude.*

2.1.3 Comparison between Form Closure and Force Closure

In Figure 2.1 both closure types can be seen in comparison. In each scenario the only external force applied is the weight force indicated by a yellow arrow, while the blue arrows show the final force direction of the contact point. As explained in Section 2.1.1, in the two-dimensional space four contact points are required. Thus the object in Figure 2.1a is safely secured with a form closure grasp. In contrast, when using the CFM, the single contacts in Figure 2.1b, are modeled with an additional tangential force onto the object, resulting into two contact forces per contact. With this property every force closure problem can be modeled as an equivalent form closure problem. This kind of equivalence is shown in Figure 2.1b, where two contacts (green) are applied on the object and their respective form closure problem is shown in blue. Once again, the object is secured because the four finger constraint is satisfied.

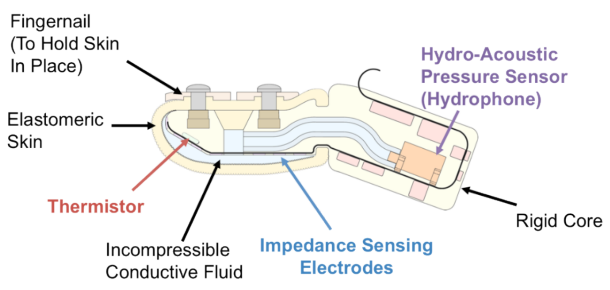
2.1.4 Tactile Sensors

On each finger of the robotic hand a tactile sensor is mounted. The sensor is called BioTac and is developed by the company Syntouch in the United States.[1] The sensoring within the finger is based on the concept of [3] while the skin of the finger goes back to [8]. A short description of this sensor is given in the following, accompanied by an illustration (Figure 2.2a) and a real example (Figure 2.2b) of the sensor in Figure 2.2 [19].
As seen in Figure 2.2a under the elastomeric skin multiple sub sensors are placed on the rigid core. Between the skin and the finger core an incompressible conductive fluid, most of the times plain salt water, is filled. The skin, protecting all sensors, is fixed with a fingernail, which can be seen in Figure 2.2b. In the finger itself three sensor that measure five different variables in total, are built in.

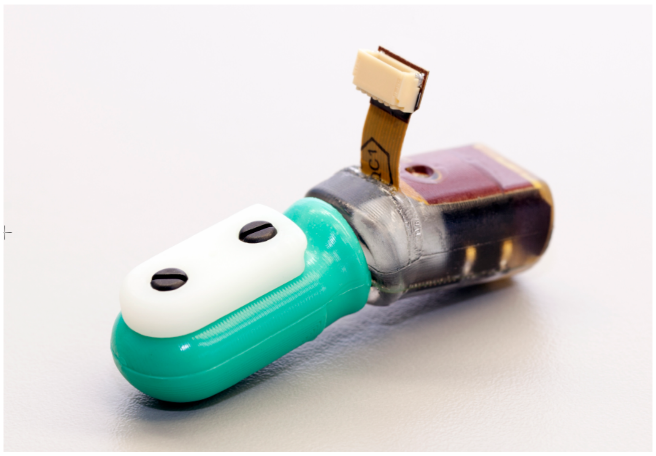
Thermistor
The thermistor is used to observe the temperature, as well as the heat flow.

Hydro-Acoustic Pressure Sensor
In the back of the BioTac sensor the hydro-acoustic pressure sensor is located. This sensor measures the dynamic fluid pressure in addition to the absolute fluid pressure. The dynamic fluid pressure is the vibration, which arises from to the non-plane skin. For measuring force the absolute fluid pressure is the most important measured quantity.

[1] Syntouch Website: https://www.syntouchinc.com/


Fingernail (To Hold Skin In Place)
Elastomeric Skin
Thermistor
Incompressible Conductive Fluid
Impedance Sensing Electrodes
Hydro-Acoustic Pressure Sensor (Hydrophone)
Rigid Core

(a) Illustration of a BioTac sensor [19]



(b) Picture of a BioTac sensor [19]

Figure 2.2.: Illustration and picture of a BioTac sensor side by side

The absolute fluid pressure P_{DC} is sampled at 100Hz while the dynamic fluid pressure P_{AC} is measured with 2200Hz. [5] used the P_{AC} to classify slip, while [4] also made use of the other sensor data to predict slip.
In fact, the absolute fluid pressure value P_{DC} is more important for predicting the force for a stable grasp. This pressure is proportional to the applied force. Hence, the applied force can be calculated from the measured P_{DC} and vice versa. Yet there is no actual interest in explicitly stating the force for this thesis. As a result, this step is left out for convenience.

Impedance Sensing Electrodes

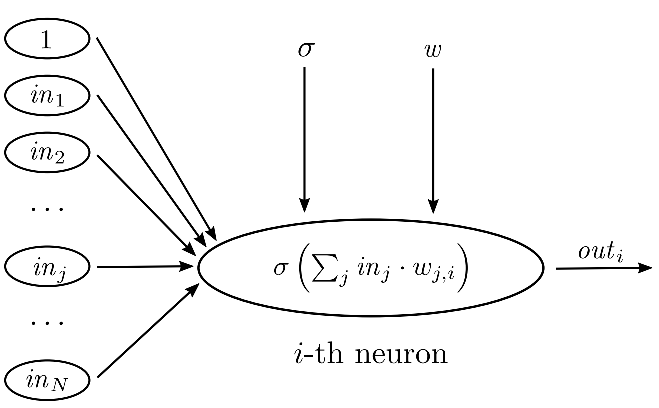
Across the whole finger tip 19 impedance sensing electrodes are spread. If the skin is deformed nearby an electrode, a change in the voltage is registered. With the voltage information of 19 electrodes in total an estimation of the normal direction of the touched surface can be made by mapping the output of these electrodes onto the three vectors forming a valid coordinate frame. It can be calculated where the object touches the finger and with the help of the pressure sensor even the force along each axis [4], [5].

2.2 Neural Networks

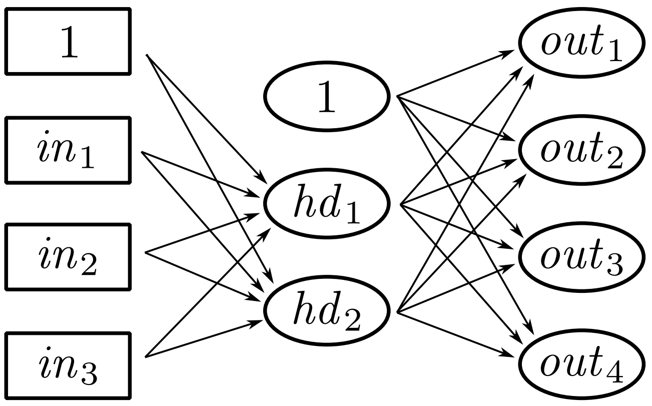
Neural Networks (NNs) are based on human brain neurons and their information processing [20]. Various types of neural networks exist, but later only the multilayer feed-forward NN is used to predict the friction coefficient. Ergo, only the concepts needed for this specific NN are explained.
As the name suggests, such a multilayer feed-forward NN consists of multiple layers. The core of such a layer are the neurons, an exemplary illustration of the i-th neuron is shown in Figure 2.3a. As an input for the neuron either the initial data or the output of the previous layer of neurons serves. In addition, a bias term – 1 – is appended to the input. In the neuron itself every input in_j is multiplied by a weight $w_{j,i}$ and summed up. The result of the sum is fed into an activation function σ of the neuron. The output out_i of the activation function is also the output of the neuron. If the neuron is in the last layer of the NN, this output is as well the output of the complete NN.
This interaction leads to a fully connected net of neurons between each layer. An example network is shown in Figure 2.3b. Each layer has a fixed amount of neurons and the additional bias term. From the three-dimensional input layer a connection to every neuron in the next layer is established. Since this layer is located between input and output, it is called a hidden layer. Once again, the additional bias term is appended. The results of each neuron in the hidden layer is passed forward to the final layer, the output layer. The amount of hidden layers and their respective amount of neurons should be adapted to the problem. As previously mentioned, for each interlayer connection a weight is associated. When assuming the first layer has a size of m, excluding the bias and the respective next layer has a size of n, also excluding the bias, the total weights needed for the connection between the layers are given by

$$(m+1)\cdot n. \tag{2.2}$$

In the learning process of a NN the initially random weights are adjusted with the help of the back propagation algorithm in an iterative process to minimize a loss function [21].



(a) i-th neuron in a Neural Network: The input of a neuron is either the data or the output of the previous layer. Inside the neuron the input is multiplied by a weight and summed up. The sum is fed into an activation function σ whose output is the output of the neuron.



(b) Example of a Neural Network: It consists of three inputs (square boxes) and the bias term. Between the output layer and the input layer a hidden layer with two neurons and one bias term is placed.

Figure 2.3.: In the left Figure the most basic element of a Neural Network is shown. Multiple neurons, which are connected as shown in the right Figure, form a layer.

2.2.1 Loss Function

Fundamentally a loss function is a metric to measure the loss between the prediction of the neural network and the real value.
Depending on the nature of the problem, which should be solved by a NN, different loss functions can be optimized.
An exemplary loss function that will be used later for a regression problem, is the Mean Squared Error (MSE). Formally defined, there are N samples to test on and for the n-th sample $y^{(n)}$ is the actual value while $\hat{y}^{(n)}$ is the prediction by the NN, the MSE is

$$\frac{1}{N}\sum_{n=0}^{N}\left(y^{(n)}-\hat{y}^{(n)}\right)^2. \tag{2.3}$$

When taking the root of this error, the resulting loss is the Root Mean Squared Error (RMSE). In comparison to MSE the RMSE is a better indicator for the actual problem because it is not squared.
If a classification problem is solved, the data is not a continuous output like in the previous regression case. One possible way is to encode the classes in a zero vector with a size of the total classes K. For the k-th true class the k-th entry of the vector is set to one. Generating this vector is called one-hot encoding. Hence, the resulting vector is an one-hot vector. The last layer of the NN consists of K output neurons and tries to reproduce this vector. If the output of the NN is scaled to sum up to one, the output can been seen as probabilities for the different classes. It follows that for classification a loss function which measures the distance between two different probability distribution is desired. The Cross Entropy (CE) which is defined by

$$-\sum_{n=0}^{N}\sum_{k=0}^{K} y_k^{(n)} \log \hat{y}_k^{(n)} \tag{2.4}$$

stands as an example for a such loss function.

2.2.2 Learning Process

As previously stated, the learning of a NN is optimizing the loss function with respect to the weights. Because the loss function might have multiple local minima, the optimization problem to solve is non-convex. For this reason, there is currently no closed form solution for an arbitrary NN. To overcome this problem iterative methods are used.
In general, the process of training can be split into two parts, the forward pass and the backward pass. In the forward pass an example is fed into the NN and passed to the end. Here the error is calculated with the help of the loss function. Based on this error, the weights for the last layer are updated. In order to update the remaining weights, the error is propagated backwards from layer to layer, where all weights are updated [21].

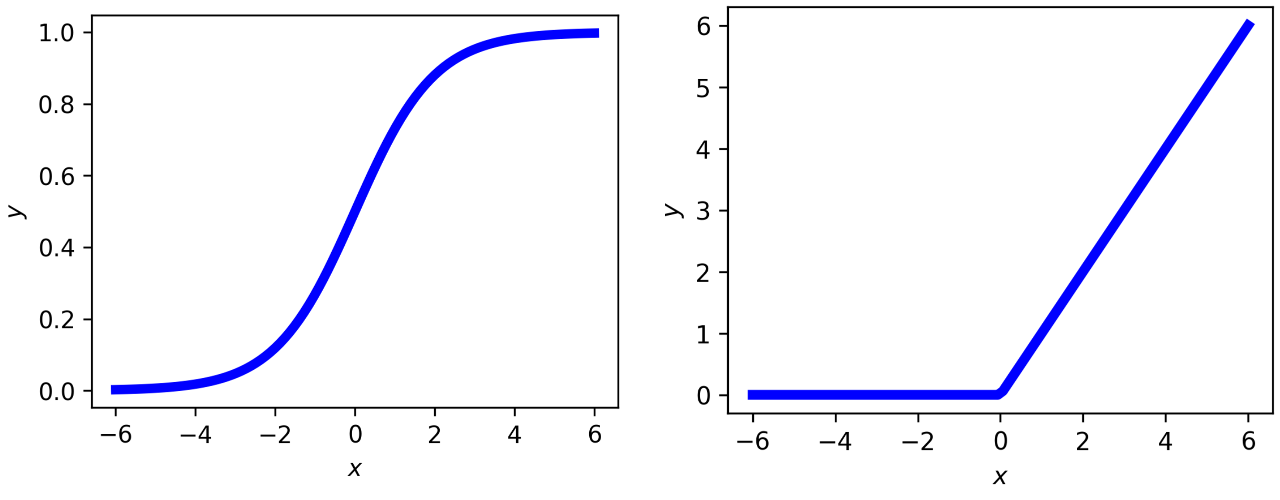
For updating the weights a simple gradient descent

$$w^{t+1} = w^t - \alpha \Delta w^t \tag{2.5}$$

could be used. In Equation 2.5 w^t denotes the current weight, Δw^t the gradient of it and α the learning rate. A more sophisticated update rule was introduced by [22]. Their algorithm Adam proved to be good. For such an algorithm multiple techniques extend the simple update step.

2.2.3 Activiation Function

One important property to remember for NNs is that, if the activation function is non-linear, it is possible to represent every non-linear continuous function, as long as there is an infinite amount of neurons in the hidden layer.
As an activation function σ various possibilities has been developed and tested. In Figure 2.4 two of the most popular choices for an activation function are shown [23].



(a) Logistic function – a sigmoid function

(b) Rectified Linear Unit

Figure 2.4.: Two popular Neural Network activation functions

Sigmoid
The first activation function – sigmoid functions – is a group of functions which form an "S". Two of the most popular functions of this group are the logistic function

$$\sigma(x) = \frac{1}{1+\exp(-x)}.$$

or the tanh function

$$\sigma(x) = \tanh(x).$$

As an example of a sigmoid function the logistic function is shown in Figure 2.4a. From now on, when a sigmoid function is used, the logistic function will be the underlying function.

Rectified Linear Unit
Another popular activation function is the Rectified Linear Unit (ReLU), which basically consists of a max operator

$$\sigma(x) = \max(0, x).$$

Compared to the sigmoid functions, ReLUs have a hardcut off at $x = 0$. [23] showed that, because of this property, training with ReLUs is several times faster in comparison to a sigmoid function. In Figure 2.4b a plot of such a ReLU is shown. However this hard cutoff comes with its drawbacks. As a matter of fact, it is not possible to compute a meaningful derivative for $x \leq 0$. As a result, the gradient is zero and the weight will not update according to Equation 2.5. To overcome this problem the concept of **leaky ReLUs** was introduced by [24], where the output is not zero but rather αx for $x \leq 0$, resulting in a gradient of α for $x \leq 0$.

2.2.4 Dropout Layers

A dropout layer randomly "drops" (setting to zero) the output of a given neuron with a probability p. The affected neuron is essentially ignored during the forward and backward pass for one sample. As a result, the NN learns not to be solely dependent on a single neuron as it might be dropped. According to [23] this behavior increases the generalization ability of the neural network because every combination of dropped neurons can be seen as a new NN. In the final network all networks are combined which can be interpreted as a form of boosting [25].

2.2.5 Convolution Layers

The last concept that is used in the final model is a convolution layer. [26] were one of the first to propose this concept of this layer. With the help of convolution layers patterns can be detected regardless of their positions in the input.
As the name suggests, the same convolution kernel with size $c \times c$ is slid over the whole multidimensional input. Typically, the follow-up step is a pooling layer in which the maximum of a small area $z \times z$ is obtained every step s on the input. When $z = s$, a non-overlapping pooling is performed, if $s < z$ the pooling is overlapping [23].
Because the convolution kernel slides over the whole multidimensional input, it has less weights than in a fully connected layer (Equation 2.2). The total amount of weights are calculated by

$$(b \cdot c^e + 1) \cdot d \tag{2.6}$$

where b is the depth of the input (e.g. RGB image: three), d the amount of convolutions applied on the multidimensional input and e the amount of sliding directions on the input (e.g. RGB image: two) as well as the obligatory bias term – 1.

2.3 Gaussian Processes

Besides NN another type of model is used – the Gaussian Process (GP) model or just GPs, which is applied on a regression problem, here GPs are mapping a continuous input $\boldsymbol{x}$ to a continuous output y defined by a function $y = f(\boldsymbol{x})$.
Assuming a simple problem like

$$\begin{aligned} t(y) &= y + \epsilon & (2.7)\\ y &= f(\boldsymbol{x}) \\ \epsilon &\sim \mathcal{N}\left(0,\ \beta^{-1}\right), & (2.8) \end{aligned}$$

where β is the precision of the unknown noise applied by the Gauss distribution $\mathcal{N}$, a simple data set consists of N pairs of $\boldsymbol{x} \in \mathscr{R}^D$ and $t \in \mathscr{R}^1$. When stacking all $\boldsymbol{x}$ in

$$\boldsymbol{X} = \begin{pmatrix} \vdots & & \vdots \\ \boldsymbol{x}^1 & \dots & \boldsymbol{x}^N \\ \vdots & & \vdots \end{pmatrix} \in \mathscr{R}^{D \times N}$$

as well as all t in the same way, the data set is formulated as one matrix and one vector.
Equation 2.7 can now be reformulated as a probability

$$p(\boldsymbol{t}|\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{t} \mid \boldsymbol{y},\ \beta^{-1}\mathbf{I}_N\right).$$

By integrating over $\boldsymbol{y}$ the marginal distribution

$$p(\boldsymbol{t}) = \int p(\boldsymbol{t}|\boldsymbol{y})p(\boldsymbol{y})\mathrm{d}\boldsymbol{y}$$

is built. By the definition of a GP, the probability $p(\boldsymbol{y})$ is expressed by

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y} \mid \mathbf{0},\ \boldsymbol{K}),$$

a Gaussian distribution $\mathcal{N}$ whose mean is zero and covariance a Gram Matrix $\boldsymbol{K}$ [21]. This matrix consists of every possible inner product between two data points in the feature space $< \boldsymbol{x}_i, \boldsymbol{x}_j >$ from all data points $\boldsymbol{X}$. Such an inner product is expressed by a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, hence, $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Now, with the help of Equation A.3 the probability for $p(\boldsymbol{t})$ is given by

$$\begin{aligned} p(\boldsymbol{t}) &= \mathcal{N}(\boldsymbol{t} \mid \mathbf{0},\ \boldsymbol{C}) \\ \text{with}\ \ \boldsymbol{C} &= \beta^{-1}\mathbf{I}_N + \boldsymbol{K}. \end{aligned}$$

Because the final goal is to make a prediction t_{N+1} for a new data point $\boldsymbol{x}_{N+1}$, it is necessary to represent the probability distribution

$$p(t_{N+1} \mid \boldsymbol{t}).$$

As a first step the joint distribution

$$p(\boldsymbol{t}_{N+1}) = \mathcal{N}(\boldsymbol{t}_{N+1} \mid \mathbf{0},\ \boldsymbol{C}_{N+1}) \tag{2.9}$$

is formed, where $\boldsymbol{t}$ is extended with t_{N+1} to $\boldsymbol{t}_{N+1}$. In a same manner the covariance matrix $\boldsymbol{C}$ is extended by

$$\boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{k} \\ \boldsymbol{k}^{\mathrm{T}} & c \end{pmatrix}. \tag{2.10}$$

In Equation 2.10 $\boldsymbol{k}$ is a vector, where each entry is the scalar product, thus, the kernel function between the new data point and every existing, formally: $\boldsymbol{k}_n = k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1})\ \forall n \in 1, \ldots, N$. Again, in a same way the scalar c is expressed by $c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$. Now the previously constructed joint distribution (Equation 2.9) can be split according to Equation A.4 and A.5 in the predictive probability distribution $\mathcal{N}\left(\mu_{\text{pred}},\ \sigma_{\text{pred}}\right)$ with parameters [21]:

$$\mu_{\text{pred}} = \boldsymbol{k}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{t} \tag{2.11}$$

$$\sigma_{\text{pred}} = c - \boldsymbol{k}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{k}. \tag{2.12}$$

In comparison to NN (Section 2.2) there is no iterative method needed, the computational process of learning consists of inverting $\boldsymbol{C}$.
The previously used kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ represents a scalar product between two data points in the feature space $< \boldsymbol{x}_i, \boldsymbol{x}_j >$. Because the kernel function expresses the similarity between these two data points, it must be symmetric. The easiest way to construct new kernels is to combine other valid kernels. In [21] multiple operations for constructing new kernels are shown. Despite that, two different kernels are presented in the following.
The **scalable Gaussian kernel**

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_0 \exp\left(-\frac{1}{2\theta_1^2} \parallel \boldsymbol{x}_i - \boldsymbol{x}_j \parallel^2\right)$$

is a simple but yet effective kernel. It only has two hyper parameters $\boldsymbol{\theta}$. θ_0 is the scaling of the kernel and θ_1 controls the bandwidth. It is even possible to drop the first hyper parameter θ_0.
Another possible kernel choice could be the **scalable Gaussian linear constant offset kernel**. As the name suggests, it is assembled from multiple smaller kernels. The complete kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_0 \exp\left(\frac{-1}{2\theta_1^2} \parallel \boldsymbol{x}_i - \boldsymbol{x}_j \parallel^2\right) + \theta_2 \boldsymbol{x}_i^T \boldsymbol{x}_j + \theta_3$$

has two more hyper parameters in comparison to the scalable Gaussian kernel. The additional parameters regulate the linear term (θ_2) as well as the constant offset (θ_3).
One way to optimize such hyper parameters is by sampling in each hyper parameter space and evaluating the error by Cross Validation (CV). In the end the hyper parameters with the lowest error should be selected. Beside the fact that this method is easy, it is not a very efficient optimization. A more sophisticated approach would use an iterative method like the gradient update described in Section 2.2.2 to move through the hyper parameter space.

Cross Validation
In CV the data set, here $\boldsymbol{X}$ as well as $\boldsymbol{y}$, is randomly split into k equally sized folds. Now, the model is trained k times on $k-1$ folds as training set and 1 fold as test set. After training the mean of the error is taken and serves as the overall error for the tested model. With CV it is possible to evaluate different models in a more general form in comparison to a fixed training and test set.

3 Problem Definition

Before a prediction based on a photo or an image *Im* can be done, the problem is, as already hinted, split into two smaller subtasks. First the derivation for this split is explained, followed by further describing the derived models.
As mentioned in Section 2.1.4, the force applied on an object is proportional to the sum of all measured static pressures $P_{\mathrm{DC}} = \sum_i P_{\mathrm{DC}}{}^{(i)}$. Formally

$$F_n = \alpha P_{\mathrm{DC}}$$

where α is a fixed scalar for all objects. Furthermore, when now assuming the CFM as the friction model, the normal force onto the object can be split into

$$F_t = \mu \cdot F_n = \mu \cdot \alpha P_{\mathrm{DC}}$$

where F_t is the tangential force. Since the thesis aims to increase the ground truth value for lifting an object, the only external applied force or moment appears due to the gravity. This force is expressed by the weight force

$$F_g = m \cdot g$$

where m is the mass of the object. To compensate this applied force it is necessary that

$$F_g = F_t, \tag{3.1}$$

hence,

$$m \cdot g = \mu \cdot \alpha P_{\mathrm{DC}}. \tag{3.2}$$

In this equation two unknown variables remain – the mass of the object m and the friction coefficient μ between the object and the finger's skin.
Assuming the mass m is given for every object, the problem can be simplified even more. Only the friction coefficient μ is left and m as well as P_{DC} is known for each object.
By separating μ from the other two variables – m and P_{DC} – a ratio

$$\frac{P_{\mathrm{DC}}}{m} = \frac{g}{\mu \cdot \alpha} = \text{Friction Coefficient Scalar} \tag{3.3}$$

between those two is given. From now on this ratio is referenced as the Friction Coefficient Scalar (FCS) for one object.
By relaxing the previously made assumption of m known, one has to predict the two variables m and FCS from an image *Im* to successful calculate the P_{dc} for an object by multiplying them. For this reason, the prediction is split into two models, one for each variable.
The first model

$$M_1(Pa) = \widehat{\mathrm{FCS}} \tag{3.4}$$

estimates the $\widehat{\mathrm{FCS}}$ for one object. The input for this model is a small patch *Pa* (e.g. 64 × 64 pixels big) from the surface of the object, extracted from the given image *Im*.
In addition to that, the second model

$$M_2(Im) = \hat{m}$$

takes the whole image *Im* as an input and predicts the mass $\hat{m}$ of the object.
The separately made predictions can be multiplied to retrieve the final pressure value

$$\hat{P}_{\mathrm{DC}} = \widehat{\mathrm{FCS}} \cdot \hat{m}.$$

Before each separate model is explained in more detail, a brief explanation of predicting the sum as well as the patch extraction is elucidated.

3.0.1 Predicting the Sum

In the very beginning the assumption was made to predict the sum over all fingers $P_{\mathrm{DC}} = \sum_i P_{\mathrm{DC}}{}^{(i)}$ and not a separate value for each finger. This approach allows a better generalization for different hand configurations. When, for instance, two fingers were used (Figure 3.1a) for surveying, let the friction coefficient be $\mu = 0.4$, and both fingers ($i = \{1,2\}$) press with a force of

$$F_n^{(i)} = 5\mathrm{N}$$

on the object, then resulting tangential force per finger is

$$F_t^{(i)} \leq 0.4 \cdot 5\mathrm{N} = 4\mathrm{N}$$

according to the CFM (Equation 2.1). When the weight force due to gravity is the only external applied force, the resulting force F_g pulling down the object is $F_g = g \cdot m$. If the object should not slip, the friction constraint (Equation 2.1)

$$F_g \leq F_t^{(1)} + F_t^{(2)} = 2\mathrm{N} \tag{3.5}$$

must be satisfied.
In another occasion, it is possible to use an additional finger on one side of the object (Figure 3.1b). To prove the made assumption that at this point it is sufficient to know the sum to prevent the object from slipping, it is important to balance the forces equally on both sides, preventing any movements v. Therefore, the single finger 1 must press with the same force like the opposite fingers 2 and 3 in sum

$$\hat{F}_n^{(1)} = \hat{F}_n^{(2)} + \hat{F}_n^{(3)}. \tag{3.6}$$

Additionally, it must be taken care of that the resulting moments are also facing opposite directions avoiding a rotation $\boldsymbol{\omega}$. When now the known sum of the previous survey

$$F_n^{(1)} + F_n^{(2)} = 10\mathrm{N}$$

is used to fulfill the constraint Equation 3.6, one possible distribution of the force might be

$$\hat{F}_n^{(1)} = 5\mathrm{N}, \qquad \hat{F}_n^{(2)} = \hat{F}_n^{(3)} = 2.5\mathrm{N}$$

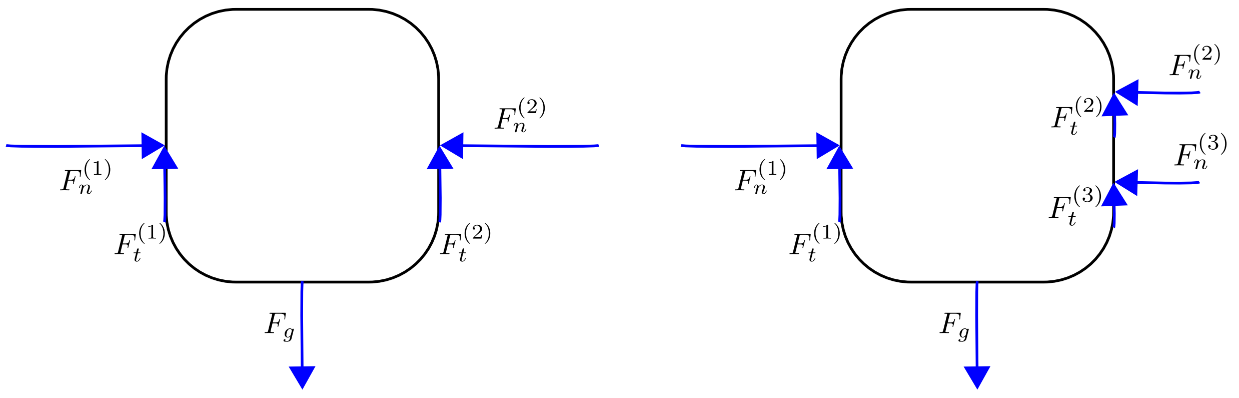
With the help of Equation 2.1 it is possible to calculate all tangential forces:

$$\hat{F}_t^{(1)} = 0.4 \cdot 5\mathrm{N} = 2\mathrm{N}, \qquad \hat{F}_t^{(2)} = \hat{F}_t^{(3)} = 0.4 \cdot 2.5\mathrm{N} = 1\mathrm{N}.$$

To prevent the object again from slipping, the friction constraint

$$F_g \leq \hat{F}_t^{(1)} + \hat{F}_t^{(2)} + \hat{F}_t^{(3)} = 2\mathrm{N} \tag{3.7}$$

must be fulfilled. As clearly seen Equation 3.5 and Equation 3.7 are equal, hence, it is shown that predicting the sum instead of every single finger value should be sufficient if the force distribution is equal on both sides along each dimension (Equation 3.6). A visualization of the problem setup is shown in Figure 3.1.



(a) In a two finger grasp forces on both sides are equally distributed, canceling out each other on the horizontal axis.

(b) Similar to the two finger grasp the forces on the horizontal axis cancel each other out in a three finger grasp.

Figure 3.1.: In order to prevent the object from slipping, the total sum of all $F_t^{(i)}$ or $\hat{F}_t^{(i)}$ must be greater than F_g. For both grasp configurations this constraint is fulfilled.

3.0.2 Patch Extraction

If an image segmentation mask is given, the extraction of a patch Pa is done straight forward. Such an image segmentation mask indicates if the corresponding pixel belongs to the object or not. The respective entry in the mask is either 0 or 1.
In each iteration n a random position (x, y) is chosen in the whole mask where the coordinates indicate the upper left corner of the possible patch Pa. With the given image segmentation mask the covered percentage for this patch C_{Pa} is calculated by simply summing over all entries in the patch Pa.
If the patch Pa covers more than a specific threshold $C_{Pa} > C_{\text{thres}}$, the patch is valid and can be used either for training or testing. The threshold is based on the past iterations n, thus, enabling to extract patches even from small objects with little coverage.

$$C_{\text{thres}} = 0.9 \exp(-0.001 \cdot n)$$

If the mask is not given for an image (e.g. an unprocessed picture), the patch extraction becomes more complicated. At this point, there are multiple possibilities, including either a construction of the image segmentation mask or a sample area must be explicitly given by the operator.

3.1 Model M_1

Like previously in Equation 3.4 defined, the model M_1 takes a small patch Pa as an input and estimates the FCS for this patch.
Because the patch is extracted from an image Im, it consists of further dimensions for representing each color in the RGB-room besides its width and height. So for an 32×32 patch the total amount of pixels are $32 \cdot 32 \cdot 3 = 3072$.
For this high-dimensional inputs NN (Section 2.2) especially Convolution Neural Networks (CNNs) were proven to be good. A CNN combines convolution layers with normal, fully connected, layers.
First the input patch is given into multiple convolution layers, which should be capable of detecting the shading on the surface of the object due to light. As previously explained in Section 2.2.5 each convolution layer is followed by a pooling layer, which reduces the size and ensures only the important information is given to the next layer.
After two convolution layers and their respective pooling operations, all resulting smaller patches are flattened and concatenated generating a single long vector. This vector is then used as an input for a fully connected NN. At this point in the NN an additional layer is inserted between the input and the output. As we want to predict the FCS, the output is a single neuron without an activation function σ. This complete network with regression output is, like in Section 2.2.2 explained, trained with the MSE as loss function.

3.2 Probabilistic Model M_1

The probabilistic equivalence for model M_1 also takes a patch Pa as input, but does not produce a single output for the FCS rather a probability distribution over the FCS dimension. To generate the one-hot vector (see Section 2.2.1) the FCS dimension is split along its axis into a fixed amount of bins B with equal size.
Now, the output of the network is a probability for each possible bin corresponding to a specific FCS instead of a single FCS output. When all outputs are normalized to sum up to one, it can be interpreted as a probability for each bin with their associated FCS.
With such knowledge the controller could first pick up the object assuming the FCS with the highest possibility. On the one hand, if a slip is occurring in this state, it could directly switch to the FCS with the second highest probability and so on. Currently each controller ([4], [5]) starts with a fixed value and must increase the pressure subsequently until the slip no longer occurs.
On the other hand, the controller could also decrease the desired pressure, when he detects no significant increase in the actual measured pressure while he tries to reach the desired pressure. This phenomena happens when the object gets deformed.

3.3 Model M_2

The second model M_2 predicts, based on an image Im, the possible mass of the object. This proposed approach also uses the image segmentation mask of the given image Im, hence, the previously provided possibilities (Section 3.0.2) remain, when no such image segmentation mask can be provided.
The underlying method used for this regression model are GPs (Section 2.3).

In order to generate a data point from its given image segmentation mask three features are extracted in a preprocessing step:

- Area a
- Height h
- Width w

The area a is calculated by counting the total area covered by the image segmentation mask, while the height h and width w are determined by measuring the distance in their respective direction between the first and last occurrence of the object in the image segmentation mask.
With these features for every image Im a quadratic feature vector

$$\boldsymbol{x}^T = \begin{pmatrix} a & h & w & a^2 & h^2 & w^2 & a \cdot h & a \cdot w & h \cdot w \end{pmatrix}$$

is constructed. This extraction assumes that the image segmentation masks were all generated from images, where the camera had the same distance to all objects. As a result, rescaling between different image segmentation masks is not needed.
The model learning and prediction follows the explained scheme in Section 2.3. Especially a set of possible hyper parameters are evaluated by using CV.

4 Experiments

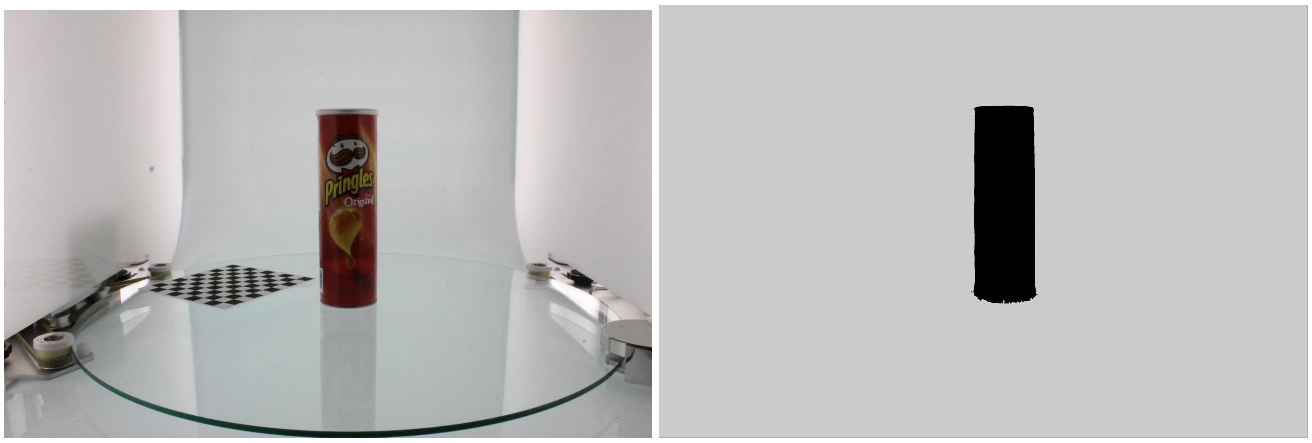
To generate the data set, the provided high resolution pictures of the objects were used, besides measuring the needed force to hold the object tight without slipping.
After surveying all objects the previously proposed models were implemented, trained and evaluated using Python.[1]

4.1 Objects

The objects are taken from the Yale–CMU–Berkeley Object Set (YCB Object set), which aims to establish a benchmark in grasping real-life objects [27]. There are a total of 104 objects in the set, but only a subset consisting of 24 objects was selected for the experiments in this thesis. All used objects and three of their important properties (mass, P_{DC} and FCS) are shown in Table A.1.
For each object

- ⊕ 600 high resolution RGB images,
- ⊖ 600 RGB-D(epth) images,
- ⊕ 600 image segmentation masks (one per each image),
- ⊖ 600 calibration parameters (one set per image),
- ⊖ five sets of textured three-dimensional geometric models,
- ⊕ and the mass

is provided [28]. Data marked with ⊕ is used for training the models and predicting the final P_{DC} from photos. In Figure 4.1 an example image (Figure 4.1a) and the respective image segmentation mask (Figure 4.1b) is shown.



(a) Exemplary image from the object data set **(b)** Exemplary image segmentation mask from the object data set

Figure 4.1.: An example image of an object with it respective image segmentation mask

The objects are photographed in five different angles ranging between 0° (full front view) and 90° (full top view). The rotating platform, which can be seen in Figure 4.1a is stopped after every 3° turn in a full 360° rotation, hence, in total $5 \cdot 120 = 600$ images per object are taken.

4.2 Robotic Setup

The previously described tactile sensors (Section 2.1.4) are mounted on a four finger robotics hand (Figure 4.2). The used robotics hand is the Allegro model by Wonik Robotics.[2]

[1] Python Website: https://www.python.org/
[2] Wonik Allegro Website: http://www.simlab.co.kr/Allegro-Hand.htm

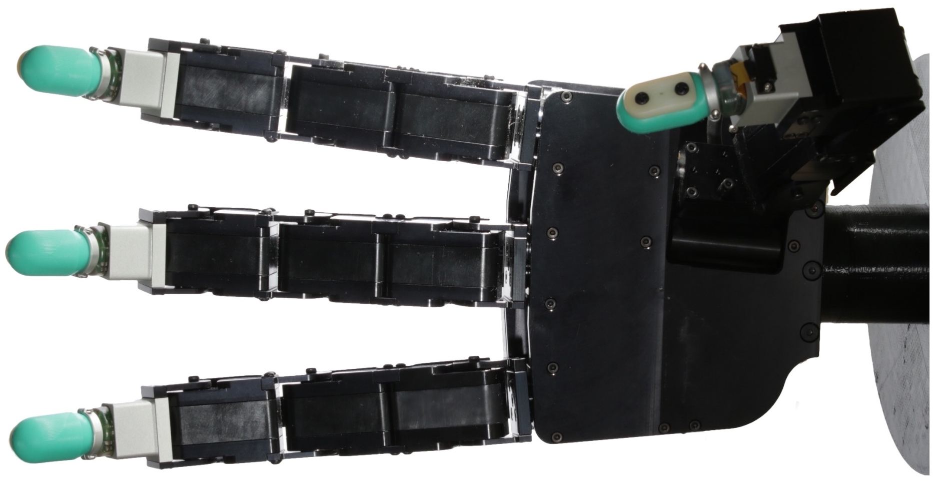


Figure 4.2.: The Allegro hand on which the tactile sensors are mounted. The hand itself is located $\approx 30\,\text{cm}$ above the table.

Each finger has four independent controllable joints, thus, there are 16 joints in total. As the hand was already used in previous experiments, it was possible to reuse the joint controller as well as the task space controller for each finger.
The hand is mounted on a wooden structure $\approx 30\,\text{cm}$ above the table to allow plenty room for slipping.
The whole setup, including the hand control and reading values of the tactile sensors, is integrated in a Robot Operating System (ROS) environment.[3] Briefly explained, as one of the core features in a ROS environment, multiple programs can communicate with each other in an asynchronous way by either topics or services. So the hand itself is controlled by various ROS topics while simultaneously the current tactile sensor values are published in their respective ROS topic independently of the controller.

4.3 Object Survey

The object survey is split into two main parts. First the object is hold by the hand, during this phase the actual measuring of the P_{DC} for this particular object is done. After multiple measurements one single P_{DC} value needs to be extracted from all of them in the second step.

4.3.1 Surveying the Objects

The rough process for surveying one object is shown in Algorithm 1.
Briefly explained, while the operator still holds the object, the hand is closed until the input P_{DC} value for every finger is reached. The operator releases the object and the current P_{DC} value is recorded for two seconds. In the end the recorded data is saved with a flag whether the object has slipped or not.
Such a survey was completed for every object between five (Figure 4.3a) to twenty (Figure 4.3b) times. In Figure 4.3 two exemplary measurements are plotted over their respective time span.
During the surveying of the objects multiple problems may occur, which either are just temporary or totally disqualifying the object from being added to the data set.
A short description of the four major problem groups is given, for a better visualization the problems are shown in Figure A.1.

1. **Fingertip sensing**
   If the normal of the surface is not correctly estimated, the finger does not move directly along the real normal onto the object, when applying the desired input P_{DC}. Such errors could result in a configuration (Figure A.1a), where the most front part of the finger touches the object. Figure 2.2a shows that, when the most front part of the finger is deformed, there is not enough fluid which gets compressed. As a result, one is not able to measure the P_{DC} sufficiently.

[3] ROS Website: http://www.ros.org/about-ros/

```
Input: $\hat{P}_{\mathrm{DC}}$ – desired value
Input: $total\_fingers$ – used fingers
Output: for each sensor – $P_{\mathrm{DC}}$ over time
// Operators holds object
open hand
move finger in contact
$reached\_fingers = 0$
while $reached\_fingers < total\_fingers$ do
    foreach $f$ in $total\_fingers$ do
        if finger_values[$f$] $> \hat{P}_{DC} + tolerance$ then
            move away from object in surface normal direction
        else if finger_values[$f$] $< \hat{P}_{DC} - tolerance$ then
            move to object in surface normal direction
        else
            $++reached\_fingers$  // one time increase per finger
// Operator releases objects
// Start survey
$start\_time$ = current_time()
while current_time() $- starttime < 2s$ do
    foreach $f$ in $total\_fingers$ do
        save($f$, finger_values[$f$])
Input: $Success$ – bool provided by operator
save($f$, $Success$)
```
Algorithm 1: Object survey

2. **Too big**
   Another similar problem occurs with too big objects, such objects may not fit in the hand (Figure A.1b). Thus, the object might touch other parts of the hand instead of the finger tips, what once again results in a wrong measure of the P_{DC}.

3. **Too heavy**
   Other objects may just be even too heavy (Figure A.1c). Due to the fine electronics inside the finger tip, no arbitrarily high force can be applied onto the object. But for heavy objects it might be necessary to apply such high force, hence, some objects can not be surveyed at all.

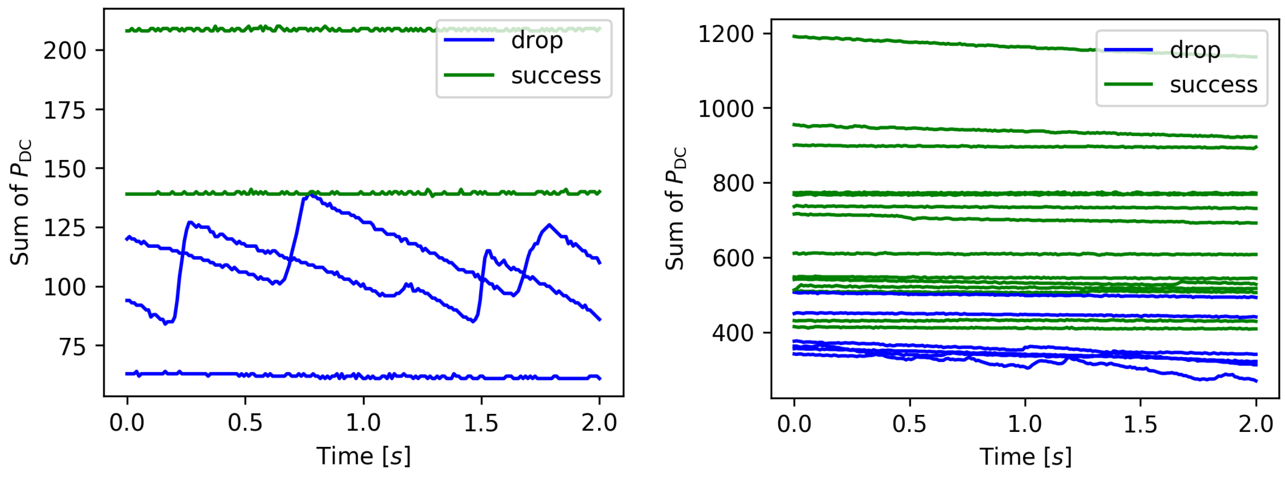
4. **Odd shape**
   The last problem also disqualifies the object from being added to the data set. In this case, the object is not too heavy, but rather has an odd shape (Figure A.1d). Therefore, the assumption that the force is always normal onto the object (Section 2.1.2) is violated.

4.3.2 Extracting the Desired P_{DC} Value

As in Section 3.0.1 described, the total sum of all measured P_{DC} values across the fingers is used instead of every single finger.
When looking at the plot of the sum of all measurements in Figure 4.3, it is clear that some form of processing needs to be done. In the end, the goal is to predict a single P_{DC} value as the ground truth for the slip controller from photos and not a continuous value over time.
The processing of the P_{DC} values is split into two steps. In the first step a single measurement is reduced to its respective mean. After that, all successful and unsuccessful measurements are convoluted to a single P_{DC} value. For extracting such a single P_{DC} value a few different methods are proposed:

1. **Min success**
   This is the most straight-forward method, the minimal P_{DC} value of all successful measurements is used.

2. **Mean of success**
   By being almost as simple as the first method, the mean of all successful trials is taken. But one need to keep in mind that the mean is always biased to where the most measurement points are surveyed.



(a) 5 measurements of the P_{DC} for the object "Spatula"

(b) 20 measurements of the P_{DC} for the object "Chips can"

Figure 4.3.: Comparison between few measurements and more

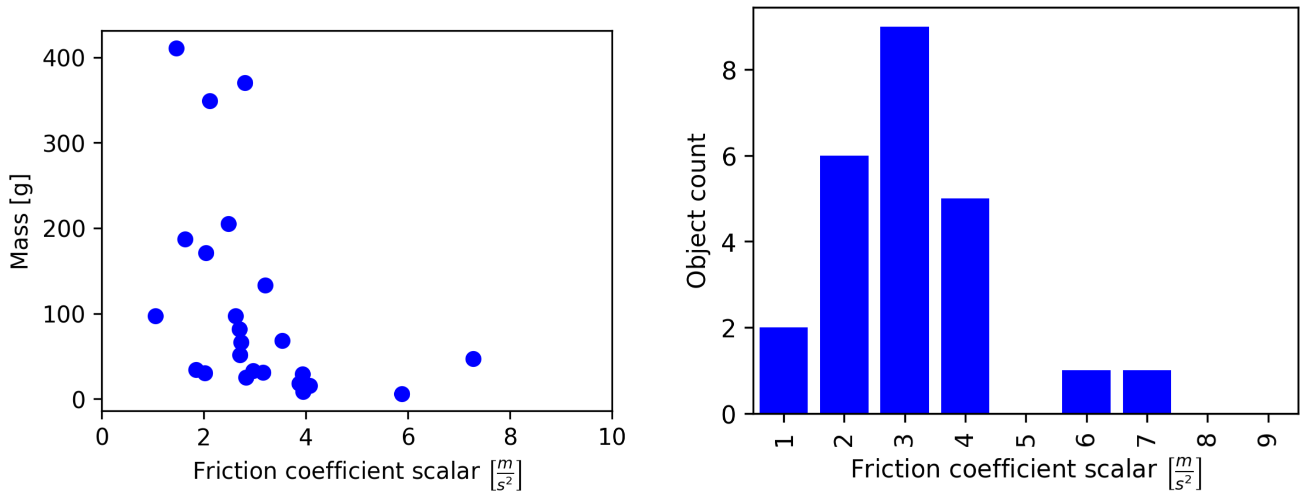
3. **Min success after highest fail**
   This method is similar to the first one. As instead of just taking the minimal successful value, the minimal successful value after the highest unsuccessful value is selected.

4. **Mean of success shifted towards mean of unsuccessful till overlap**
   The last method is a little bit more sophisticated by combining the second and the third method. Once again the mean of the successful trials is taken as well as of the unsuccessful ones and additionally their respective variance. Based on these two Gaussian distributions, the mean of the successful trials is shifted towards the mean of the unsuccessful ones. The goal is reached when the total overlapping area has reached a user defined minimum. At this stage the shifted mean of the successful trials serves as the final P_{DC}.

Because data was limited for some objects (5 trials – Figure 4.3a), the third method was selected for this thesis.

4.3.3 Final Data Distribution

With the now extracted P_{DC} value and the respective mass per object the FCS for each object can be determined with the help of Equation 3.3. The distribution of the objects is shown in Figure 4.4a. In this Figure both variables to predict – mass and FCS – are plotted. Each blue dot represents an object with its respective mass and FCS. As it can be seen in the plot, the objects are not equally distributed across both dimensions. As seen on the vertical axis, most of the objects weigh between 0 g and 100 g, while the FCS values could be assumed to be Gauss distributed. In Figure 4.4b this Gaussian distribution of all FCS is abstracted visibly. The histogram was generated with a bin width of 1 and centers at every full step $1, \ldots, 9$. From the histogram the mean of all FCS can be determined to ≈ 3.



(a) The final distribution of the mass as well as the FCS for each object (blue dots): The objects are not evenly spread across the whole dimension. Most of the objects weigh from $0\,\mathrm{g}$ to $100\,\mathrm{g}$ and have a FCS between $2\,\mathrm{m/s^2}$ and $4\,\mathrm{m/s^2}$.

(b) By sorting the objects in bins with width 1 and centers at every full step $1,\ldots,9$ the following histogram results. As it already could be guessed from the distribution in the left Figure 4.4a, most of the objects reside in the bins $2, 3$ and 4.

Figure 4.4.: Scatter plot to show the two predictive dimensions as well as the histogram for the FCS

4.4 Model Learning

Both models were implemented in Python.[4] As previously (Section 4.2) described, the communication with the hand is done in a ROS environment using the ROS Python Application Programming Interface (API). For learning the NNs the framework Keras[5] was used, which acts as a high level API for either Theano[6] or Tensorflow[7], both are opensource NN libraries.

In order to train M_1 as well as its probabilistic equivalent, small patches Pa are needed. The extraction of such patches is done like in Section 3.0.2 described. Per object and per image are in total 11 patches extracted, but only from the subset of images which directly point to the object. This subset consists of the both lower camera angles, hence, a total of $2 \cdot 120 = 240$ images. Resulting in a total of

$$2 \cdot 120 \cdot 11 = 2640$$

patches per object, thus, in a data set with

$$2640 \cdot 24 = 63360$$

patches of a size of either 32×32 or 64×64.

4.4.1 Model M_1

Based on the previous explained concepts of NNs, it is possible to refine the rough structure for M_1 introduced in Section 3.1. Different variations of the already proposed network were tested. Statistics for all different models can be found in Table 4.1. As an example the baseline model is shown in Figure A.2a with its two main components, first the convolution layers, followed by a fully connected NN with two hidden layers.

The first step – two convolution layers (Section 2.2.5) – reduces the thousands of pixel of the input patch Pa to a reasonable number. In the first layer eight bigger convolution kernels of size 5×5 are used, followed by a pooling of size and step width two. When assuming an input patch of size 32×32, the convolution kernel stops before it overlaps the border of the patch resulting in a reduction of two pixels on each side of the image. With the help of Equation 2.2 the total weights for this layer are $\left(3 \cdot 5^2 + 1\right) \cdot 8 = 608$.

[4] Python Website: https://www.python.org/
[5] Keras Website: https://keras.io/
[6] Theano Website: http://deeplearning.net/software/theano/
[7] Tensorflow Website: https://www.tensorflow.org/

Table 4.1.: All tested network configurations for model M_1

Name	Patch size	32 × 32			64 × 64		
	Structure	Weights	Conv.	MSE	Weights	Conv.	MSE
Baseline	A.2a	6,085	A.4a	1.876	26,565	A.4b	1.830
2.	Baseline + Added ReLU to last layer	6,085	A.4c	1.868	26,565	A.4d	1.830
3.	2. + Double convolutions per layer	12,169	A.4e	1.872	53,129	A.4f	11.275
4.	2. + Add a third layer + increase first layer size	12,805	A.4g	2.141	53,765	A.4h	1.908
5.	2. + Sigmoid instead of ReLU in NN	6,085	A.4i	**1.851**	26,565	A.4j	1.829
6.	5. + Overlapping pooling + increase first layer size	8.421	A.4i	1.868	45,285	A.4l	**1.817**

With a pooling step of width two the convoluted patch of size 28×28 gets reduced to $28/2 = 14$ in each dimension. Because there are eight convolutions in this layer, the new "patch" has a new depth of size eight. In the next convolution layer the size of the convolution kernel is reduced to 3×3 and only four kernels with this size are applied. Once again the total weights are given by Equation 2.6: $\left(8 \cdot 3^2 + 1\right) \cdot 4 = 292$. Here also a pooling takes place to reduce the dimensions of the patch even more, resulting in a total size of $6 \times 6 \times 4$. But before moving to the next step, the patch is flatted into an one-dimensional vector with size $6 \cdot 6 \cdot 4 = 144$.
This output is used in the second step – two fully connected layers – to solve a regression problem. The first hidden layer has a size of 32. With Equation 2.2 given, the amount weights for the first hidden layer is $(144 + 1) \cdot 32 = 4640$. For this layer the dropout mechanism (Section 2.2.4) has been used. The following hidden layer has a size of 16 resulting in $(32 + 1) \cdot 16 = 528$ weights according to Equation 2.2. Because a regression output is wanted the last layer only has one neuron, hence, the amount of weights for the last connection is given by $(16 + 1) \cdot 1 = 17$.
The resulting sum of all weights is

$$608 + 292 + 4640 + 528 + 17 = 6085.$$

In this baseline model all activation functions σ are ReLUs.
Before the training of the model can begin, the data set is split into two parts, the test set and the training set. With 24 objects in total, 20 objects were randomly assigned to the training set and the four remaining objects form the test set. This separation is kept from now on equal for all models. After the split the models were trained with the MSE (Equation 2.3) as their loss function for 100 epochs, in each epoch every trainings patch is used exactly once. After each epoch the current model is evaluated on the test set. If the current model is better than the previous best model, the current model is saved as the new current best model. The progression over time for each model is illustrated in Figure A.4. After the 100 epochs of training the best model is tested on the whole data set and the loss collected in Table 4.1.
In the evolution of the training and test error (Figure A.4) it can be seen that the error on the test does not further decrease but rather the network oscillates between two possible minima. To prove this assumption the 2. network structure with a patch size of 32×32 was trained for 1000 epochs. The resulting evolution is shown in Figure 4.5. In this plot the oscillations can be identified to start after the 100 epochs, therefore, a 100 epochs were chosen for training all models.

4.4.2 Probabilistic Model M_1

As in Section 3.2 outlined, the distribution over all FCS needs to be discretized in multiple bins. As previously shown in Figure 4.4b the data resides naturally in some bins especially at the positions 2, 3 and 4. This histogram serves as the baseline for constructing the one-hot vector. Based on it the distribution is divided in B equally sized bins with a size of 1 with centers at $1, 2, \ldots 9$. As in Section 2.2.2 explained in order to handle a class output instead of the MSE as a loss function the model is trained and evaluated using CE (Equation 2.4). In addition to CE, a more interpretable metric, the top k accuracy metric is now introduced. This metric checks if the actual class is among the k highest predicted classes, if so, the test is counted as successful. For multiple training examples the metric calculates the percentage of successful tests from all tests. Besides the CE all models were also evaluated using this metric with $k = 3$ as its parameter. Even the

model selection for the probabilistic M_1 is not based on CE as its loss function, but rather on the top 3 accuracy. A special case holds for $k = 1$, where it is the normal accuracy.
The network structure and changes (Table 4.2) are similar to Model M_1 in Section 4.4.1. As for the non-probabilistic model the bigger patches of size 64×64 do not show any significant benefits and take more time to train the model, the probabilistic model M_1 is only trained on the smaller patches of size 32×32.

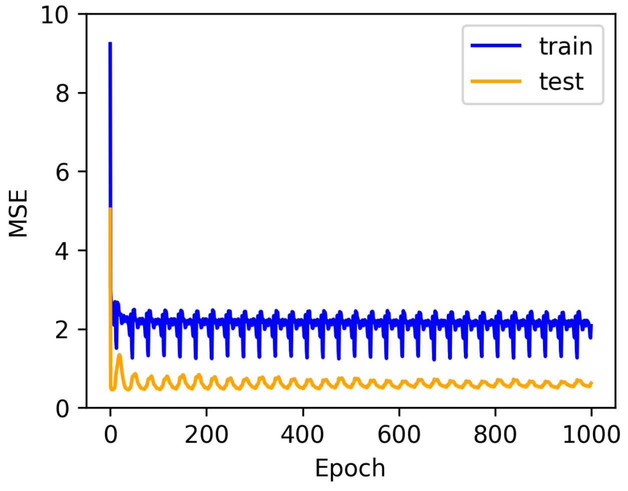


Figure 4.5.: 1000 epochs of training for model M_1 with a patch size of 32×32 to show the oscillation between two minima

Table 4.2.: All tested network configurations for the probabilistic model M_1

Name	Patch size	**32×32**	
	Structure	Weights	Top 3 acc.
Baseline	A.2b	6,305	0.8275
2.	Baseline + Increased size of first hidden layer	11,521	0.8279
3.	Baseline + Double convolutions per layer	12,389	0.8284
4.	2. + Add a third layer	13,025	0.8274
5.	Baseline + Sigmoid instead of ReLU in NN	6,305	0.8280
6.	3. + Overlapping pooling + increase first layer size	9,573	**0.8285**

4.4.3 Model M_2

As in 3.3 proposed, for the second model M_2 GPs with different kernels were tested. In addition to that a comparison was made to Bayesian linear regression.[8]
Because training on the whole data set of images $24 \cdot 120 = 2880$ is too expensive computation-wise, it was tested if training on less data actually would impair the result. For testing this hypotheses the models were learned on 1%, 5% and 10% of the data set.
Having already explained, the hyper parameters were optimized by sampling each parameter and combining them in every possible combination. Such a combination of hyper parameters is tested using 6 fold CV. For example, when there are two hyper parameters, each sampled ten times, $10^2 \cdot 6 = 600$ models are learned and evaluated. One important thing to notice is that the precision β of the underlying noise distribution (Equation 2.8) is not known, hence, it will account into the hyper parameters as well.
Before the models can be tested, it is important to standardize the data. Naturally the area covered a of an image segmentation mask is equal to the height h or width w squared, ergo, multiple times bigger. Standardizing the data leads to a more robust result in the end. Before the model is learned every data point of the training set, consisting of $\boldsymbol{x}$ and t, is modified by the mean $\bar{x}^{(i)}$ of the training set as well as standard deviation $s^{(i)}$ along each dimension $x^{(i)}$ including the target t:

$$\tilde{x}^{(i)} = \frac{x^{(i)} - \bar{x}^{(i)}}{s^{(i)}}, \quad \tilde{t}^{(i)} = \frac{t - \bar{t}}{s^{(t)}}.$$

The resulting standardized data set is then used to train multiple models and evaluating them.

[8] For information about Bayesian linear regression please see [21]

A new data point is standardized with the mean $\bar{x}^{(i)}$ and variance $s^{(i)}$ of the training set. After a prediction $\hat{t}$ is made, the target value t is recovered with the help of the target mean of the training set $\bar{t}$ as well as the standard deviation $s^{(t)}$

$$t = \hat{t} \cdot s^{(t)} + \bar{t}.$$

An overview over all tested models is shown in Table 4.3. Because the mass is again an regression output, the MSE is used to select the best model, and the RMSE is presented for a more interpretable result. Because the MSE is still squared, the error can not be compared to the absolute value, while with the RMSE it is possible.
Unfortunately, the scalable Gaussian linear constant offset kernel could not further be explored due to the sheer amount of hyper parameters needed to be sampled.

Table 4.3.: All tested kernels for Gaussian Processes

Method/Kernel	Tested hyper parameters	Percentage used [%]	Best RMSE
Bayesian linear regression	10	1	122.89
	10	5	132.35
	10	10	132.40
GPs with Gauss kernel	100	1	131.03
	100	5	129.45
	100	10	131.20
GPs with linear kernel, no offset	1600	1	201.77
	400	5	193.76
	400	10	190.38
GPs with scalable Gauss kernel	1000	1	130.97
	343	5	129.38
	125	10	131.20
GPs with scalable Gaussian linear constant offset kernel	3125	1	96.079
	1024	5	140.157
	–	10	–

5 Discussion

Two major assumptions were made. On the one side it was assumed that the perfect image segmentation mask is given and on the other side that the mass is known.
The assumption that the perfect image segmentation mask is given is not the regular case when grasping an object in a clustered environment. In such a scenario the object might be only partly visible due to other objects covering it. As a result, for extracting the patch and determining the area a, height h and width w of the object more sophisticated approaches are needed.
When the assumption is kept that the mass is known, the prediction solely relies on M_1 or the probabilistic equivalent. As the P_{DC} prediction is only depending on one model, the final prediction is less error-prone.
Nonetheless, to answer the question whether it is possible to predict grasp forces or not, it is, but currently not on point with the proposed methods. In the following it is shown and in depth discussed, where still problems remain to be solved.

5.1 Predicting the Friction Coefficient Scalar

First the prediction of the continuous output is discussed, second the prediction of the probabilistic equivalent is evaluated.

5.1.1 Model M_1

As marked in Table 4.1, the best MSE is 1.851 (32×32 patches) or 1.817 (64×64 patches). When taking the root of both MSE, the resulting RMSE for the FCS is ≈ 1.35. As seen in Figure 4.4a, the absolute values goes up to 7.2, resulting in a relative mean error of $1.35/7.2 \approx 18.75\%$. Given such a value, there is clearly room for improvement (see Chapter 6).
Nonetheless, the slight changes in the structure of the NN did not have any significant impact on the MSE. In Figure A.4 the loss evolution for all different models is even similar, as the evolution of the training as well as the test loss follows the same pattern throughout every model. Additionally as already mentioned in Section 4.4.1, the training of the model is periodic, resulting in no further decrease of the loss with increasing amount epochs learned.
Despite that, in Table 4.1 is shown that the usage of the bigger patch size slightly increases the performance of the network, even though the additional weights. But with such a bigger patch size from 32×32 to 64×64 the overhead of extracting and storing increases exponentially with two as exponent. The increasing amount of weights needed for the same structure also has its downsides. As to keep the amount weights as little as possible, less complex structures are possible for 64×64 patches instead of size 32×32. As seen in Table 4.1, when the amount of weights trained is near the used training patches (63360), the MSE increases moderately (4. model) or even drastically (3. model).
Nonetheless, in Figure A.4l the loss for the best model is plotted over the total epochs trained. For this model the overlapping pooling technique (Section 2.2.5) was used for the convolution layers. For each pooling operation the maximum of a 3×3 square is taken every second entry. In addition to the overlapping pooling the first layer in the fully connected NN used the logistic function as its activation function σ, as well as the size of the first hidden layer was increased to 64. As in the beginning mentioned, the total RMSE for this model is ≈ 1.35. The model itself is taken from the 84. epoch, where the minimum of the loss on the test set was achieved.

5.1.2 Probabilistic Model M_1

The probabilistic equivalent to model M_1 shares some characteristics with to the non-probabilistic model. Interesting though is the non-decreasing CE of the test set across all trained models after the first or second epoch trained. This is an indicator for overfitting – the learned model does not generalize well to unseen data. To overcome overfitting before it even occurs, an additional drop out layer was inserted after the first hidden layer. Another tested technique was to randomly leave out examples during training. Hence, per epoch not the complete data set was utilized for training but rather only e.g. 80%, based on the leave out probability. In the long run this technique did not solve the problem of early overfitting, but rather postponed the effect to the 5th or 6th epoch.
Nonetheless, with or without applied techniques during training the top 3 accuracy on the test set went up, resulting in a better prediction in the end. Due to a bin width of 1 the mean error which naturally occurs when sorting in the correct

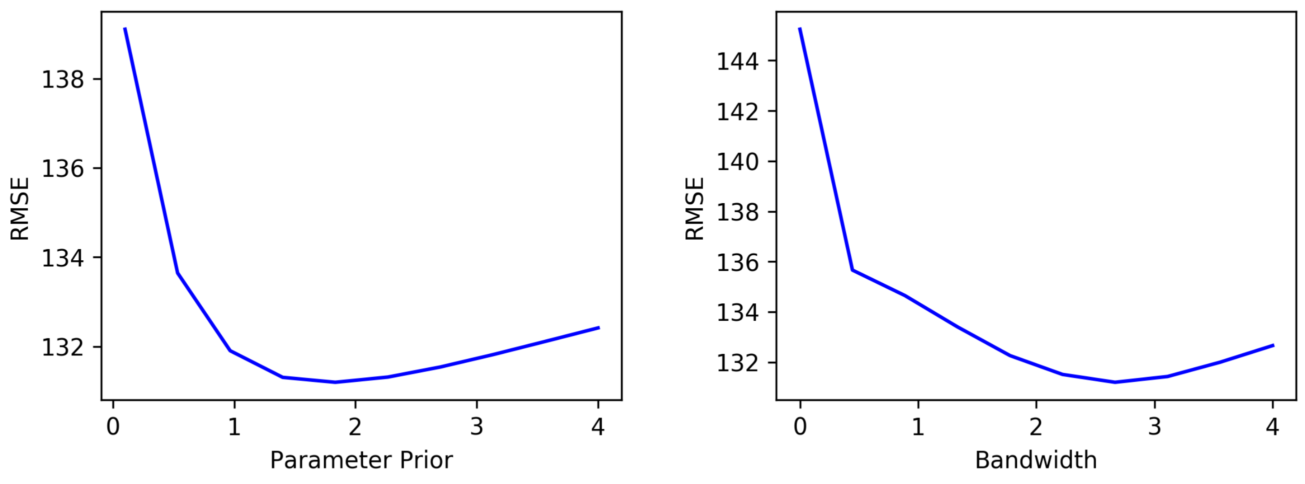
bin is at max 0.5. When the wrong bin is predicted, the error increases by the bin width every bin apart. For example, when the actual value is in the adjacent bin to the predicted bin, the maximum error is 1.5. So the non-probabilistic model (best RMSE: 1.35) would predict in the average one bin off. Based on this connection the top 3 accuracy was chosen to allow the probabilist model to cover the same area as the non-probabilistic model.
Because the models trained for the probabilistic M_1 are similar to the non-probabilistic equivalent, the variations of the network had as well no significant impact, the model was only slightly improved. One significant thing all models shared, is the increase of the top 3 accuracy from roughly 0.74 to 0.77. Like in the non-probabilistic model the evolution of the respective loss function is similar across all trained models.
As the increasing patch size had no significant improvements for the non-probabilistic model, it was decided to test the probabilistic model only on 32×32 patches. The best model achieves an overall top 3 accuracy of 0.8285.
[5] showed that their algorithm is feasible to detect a slip within 30 ms before an inertial measurement unit could detect it, while [4] even stated to detect slip up to 200 ms in the future. Assuming the robotic hand can increase the applied force within in a reasonable time, the hand should be able to test and apply the top 3 predicted FCS before the objects fully slip out of the hand. As seen in Table 4.2, the best model trained was 6., predicting the correct class along the top 3 predicted classes with a success rate of 82.85%.

5.2 Predicting the Mass – Model M_2

Regarding predicting the mass, the smallest RMSE achieved was ≈ 130 by using the (scalable) Gaussian kernel. As a result, the following hyper parameters were optimized:

- parameter prior
- kernel bandwidth
- (and scaling factor)

Assuming such a RMSE, it is clear that the mass prediction is not perfect, but there is some form of correlation between the defined features and the mass. As in the very beginning mentioned, the rather high RMSE is a result of the unknown density and unknown depth of the possible object. With the help of the depth the total volume can be estimated more accurately, multiplied by the density a better mass prediction should be made.
When comparing this result to the other kernels tried, it can be seen that the other kernels were good as well, but had more hyper parameters. Due to the inefficient method of hyper parameter optimization, it is easier to use a more simple kernel like the non-scalable Gaussian kernel. As explained previously, the sheer amount of hyper parameters for the scalable Gaussian linear constant offset kernel made it impossible to find the perfect set of hyper parameters for this kernel.
The assumption that decreasing the size of the data set does not yield worse results, were also proved to be true. Across all different kernels it was shown that the decrease training set size does not significantly increase the RMSE and even decreases it for some configuration. As the testing of one hyper parameter configuration on 10% instead of 1% of the data set takes roughly 100 times longer with 6 fold CV, training on the smaller data set significantly increases the speed of training. Due to many hyper parameter configurations across all kernels it is useful to identify a range for each hyper parameter first, in which the best value for this hyper parameter is residing. By identifying the range on 1% of the data set beforehand and finally only optimizing the hyper parameters within the given range on 10% of the data set, the inefficient way of selecting the hyper parameters could fasten up a little bit. The medium sized data set of 5% can be used to refine the rough range.
As this approach of hyper parameter optimization is capped by the amount of samples for each hyper parameter exponential to the total hyper parameters, a better approach at this point would be, like previously already mentioned, making use of a gradient-based method (Equation 2.5). For each hyper parameter it is possible to follow the gradients direction of the steepest descent in its respective dimensions, following this direction reduces the massive amount of unnecessary sampling of useless hyper parameters. Even so, for such a gradient-based method once again hyper parameters are needed, like the learning rate. This learning rate could be optimized again. Following down this road, leads to an infinite learning of the previous hyper parameters. For this reason the used method by first searching a range, which serves as the baseline, on the smaller data set – 1% – then optimizing for the bigger data set – 10% – within the range, is sufficient for a small amount of hyper parameters (up to 3).
However, in Figure 5.1 the hyper parameter optimization for the normal Gauss kernel on 10% of the data set is shown. As seen in Figure 5.1a, the parameter prior was optimized between zero and four. This range was, as previously explained, determined on only 1% of the data set. The resulting parameter prior on 10% is ≈ 1.833 while on 1% of the data set it is ≈ 0.716. Similar small differences apply for the kernel bandwidth as well. For only two hyper parameters the connection between those are relatively easy to decouple and can be optimized with plain sampling. With the increasing amount



(a) RMSE over the parameter prior

(b) RMSE over the kernel bandwidth

Figure 5.1.: Evolution of the RMSE over the hyper parameter dimensions: The plots are generated using 10% of the data set, while beforehand the range in within the hyper parameter are may residing is determined only on 1% of the data set.

of hyper parameters for the scalable Gaussian linear constant offset kernel it seems nearly impossible to determine the range in within the best hyper parameters are residing because the ranges for each hyper parameter affect each other heavily.
When comparing the choice of GPs over Bayesian Linear regression in the first place, it seems that there is no big improvement. The RMSE on the bigger data sets – 5% and 10% – for both are in an equal domain. One thing to notice is that in Bayesian Linear regression the amount of data points actually matters. For example, GPs with a plain Gaussian kernel show similar results in the RMSE, not depending on the amount of selected data points. Besides that difference, the hyper parameters for Bayesian Linear regression needed to be drastically readjusted depending on the size of the data set. While for the GPs the hyper parameters could be kept almost constant for each kernel across the three tested size of the data set. Obviously best values for the hyper parameters will vary, but the minima remain within in the same range, when adding more data points to set. Such a different behavior for Bayesian Linear regression is not necessarily bad, but will dampen increasing the data set size.
Looking at a completely different direction, a possibility might be not to extract features (area, width and height) in the first place. Here experiments have shown that the resulting dimensions of 720000, with either zero or one as an entry, is not feasible.

5.3 Theoretical Evaluation

One problem that naturally occurs due to the selection of random patches, is the non-optimality of some training patches. Such patches may not inhabit the important patterns to predict the FCS successfully. On the other side they act as a generalization trade off because they add noise to the training set.
Another problem which only concerns the non-probabilistic model, is the not existent measurement for uncertainty of the prediction. With the help of a known uncertainty for the made prediction it is possible to predict from another patch, if the uncertainty is higher than a specific value. This is already possible for the probabilistic equivalent because the probability for each class is given. If there are no significant spikes in the probabilities, another prediction with a different patch can be made. Because this could happen before the hand actual touches the object, the re-predicting is done off-line and could be repeated arbitrarily times.
As a next step the actual output was observed instead of just comparing the RMSEs and the top 3 accuracy with an interesting result. The neural network mostly predicts the same value of 3.117 (Baseline NN, 32×32 patches). The reason for such a behavior is rooted in too much noise in the data. The previously as positive described effect of non-optimal patches has its downsides in the long run. As a result, the networks do not make a prediction based on the patch but rather act as a more sophisticated approximation of the mean for all training examples. Despite of that prediction, such a behavior is not wanted but could be expected. When looking at Figure 4.4b, the distribution of the FCS could be approximated with a Gaussian distribution with a mean of the predicted value 3.117. Knowing this faulty prediction, it could easily be explained why the changes to the networks had no big impact, as the changes just adjusted a little bit the behavior of how the mean of the distribution is approximated.

As predicting the FCS is only one half towards the goal, the mass prediction needs to be tested as well. Here, not only the plain output of a single image segmentation mask was used, but rather the output of multiple masks combined. The mass for 100 masks in total were predicted by Equation 2.11, while the variance (Equation 2.12) was recorded as well. With the help of the lowest variance the ten best predictions were selected of the 100 in total. The variance gauges the uncertainty, hence, a low variance indicates a more certain prediction. In comparison to the FCS actually different masses are predicted, as far as the correlation between input and output allows it, the limiting factors seem to be the unknown density and the unknown volume.

5.4 Improved Photos

The question, which raises at this point, is, if the selection of patches could be improved, the NN could actually learn a correlation between a patch and its respective FCS. To prove this, new pictures of the object were taken instead of using the provided photos from the YCB Object set. The setup to take these pictures is shown in Figure 5.2. One important difference in comparison to the provided images is the presence of a point light instead of a diffuse light source like in the provided images. This new light source should enhance the surface of the objects more than with a diffuse light source. Because the process could not be automated, only 10 to 15 high resolution photos per object were taken, 285 photos in total.

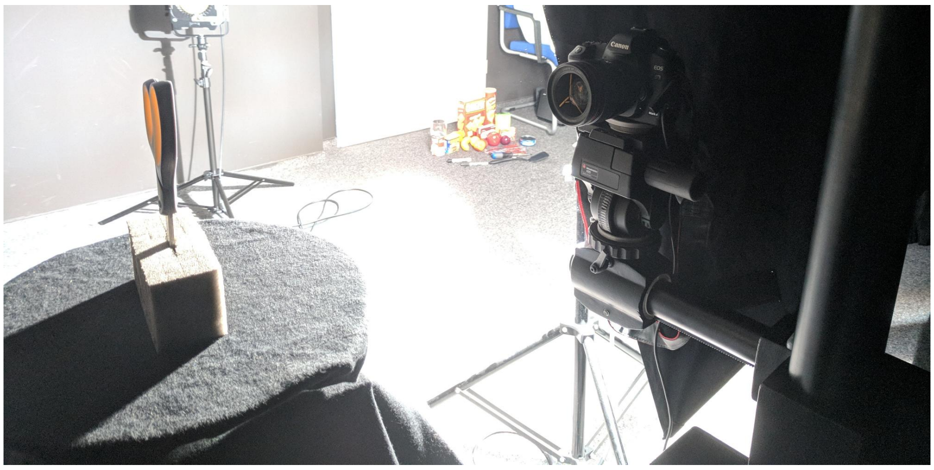


Figure 5.2.: Setup for taking the improved images: The object is placed on a turntable (left) allowing to rotate it 360°. With the black curtain (right) the point light source is covered to reduce possible lens flare in the photo.

5.4.1 Patch Extraction

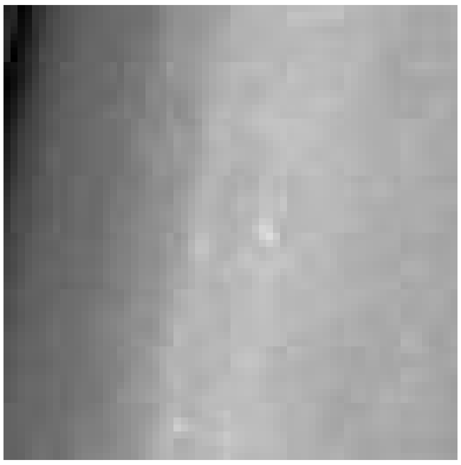
As previously mentioned (Section 3.0.2) the extraction from unprocessed images without a given image segmentation mask is more complex. To solve this problem a rectangular area in every photo is selected by the operator, in which the patch should be extracted for this specific image. After the rough area selection once again random patches are sampled across this whole area. The possible patch is then transformed into gray scale. After the transformation the area of near black pixels are calculated by thresholding all pixels. If the amount of near black pixels exceeds a given percentage, a new patch will be sampled from the operator-defined area. For each operator-defined area 102 patches are extracted ultimately leading to

$$285 \cdot 102 = 29070$$

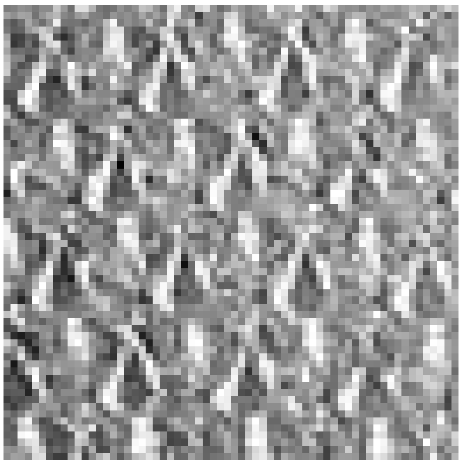
patches. In order not to bias the NN towards the more photographed objects, a subset of images per object is selected with the size of the least provided images $10 \cdot 102 = 1020$ per object, a total data set size of

$$24 \cdot 1020 = 24480$$

photos was generated.



(a) Smooth surface patch, from the object "Spoon": The corresponding FCS was determined to 2.014.



(b) Rough surface patch, from the object "Sponge": The corresponding FCS was determined to 5.88.

Figure 5.3.: Comparison between a smooth and a rough surface patch: The FCS for the smooth surface patch is lower in comparison to the rough surface patch.

5.4.2 Patch Transformation

In order to enrich the surface of the patch, multiple transformations were applied on the image. As a first step, a bigger patch of size 128 × 128 is sampled from the operator-defined area instead of the final patch size. Remembering at this point, the patch is a gray scale, the bigger patch is normalized. The normalization enhances the surface even more by increasing highlights and decreasing darker areas. As the last step the patch is downscaled with a bicubic interpolation to the desired patch size of either 16 × 16, 32 × 32 or 64 × 64 for training the network. Extracting a bigger patch and downsampling is needed because in some cases an initial smaller patch would not capture enough of the surface.

Two resulting exemplary patches with their respective FCS are shown in Figure 5.3. On the left side (Figure 5.3a) the surface is smoother in comparison to the rougher surface on the right. Therefore, the left object has a lower FCS, while the FCS for the right one (Figure 5.3b) is obviously higher. The NN should recognize this property as well and predict a higher FCS, if a rough surface patch is given.

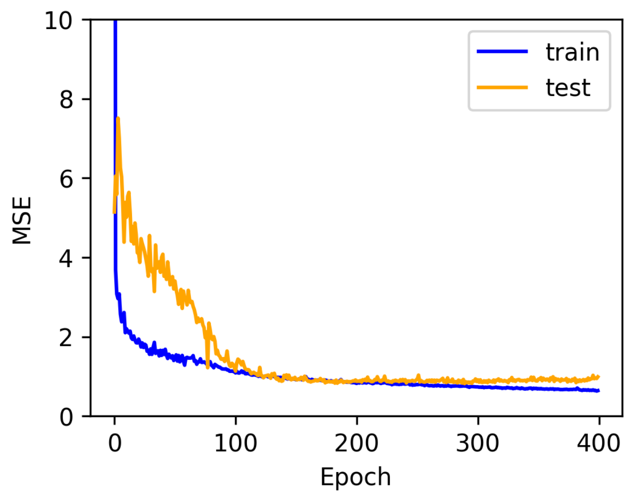
5.4.3 Retraining the Neural Networks

Before the NNs are retrained, the FCS for the object "Knife" and "Orange" were modified by hand. Because the knife, has the same material like the spoon and the fork, accordingly its value was changed from 3.15 to 2.07. Similar procedure for the orange, its surface is a bit smoother than the lemon but not as smooth as the banana's surface. Hence, the FCS was changed from 7.28 to 3.09. As a result, the distribution of all FCS became even thinner. The bins were adapted by reducing the amount to six bins with centers at $1, \ldots, 6$, but still with a width of 1.

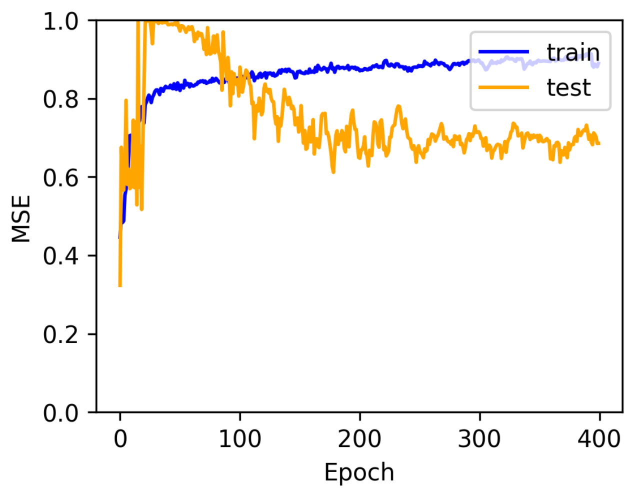
Due to the change of the data set, the network structure was adapted to it. As now gray scale patches are used, the depth of one patch is reduced from three (RGB) to one. Because of the reduction of the input size and the resulting less weights, the first convolution layer was increased in its kernel size and amount of kernels applied, as well as the padding was changed to mirror instead of cutting of. In addition to these changes, a third convolution layer was added. For all convolution layers overlapping pooling was still used. The first hidden layer size was increased from 32 to 96 nodes, while the dropout mechanism was kept. Instead of plain ReLUs, leaky ReLUs were used as the activation function across each layer. The complete structure for the network (Figure A.3a) as well as the probabilistic output (Figure A.3b) is shown in Figure A.3.

The new NN structures for the non-probabilistic model as well as the probabilistic one were trained on the new patches. As before, the training on bigger patches (64 × 64 instead of 32 × 32) made no remarkable differences because both patches were initially sampled with a size of 128 × 128 before being downscaled to their desired size. With patches sampled at a size of 96 × 96 downscaled to 16 × 16, the error increased. Therefore, the network structure is trained on patches of a size of 32 × 32.

After training the best RMSE on the complete data set for the non-probabilistic model was 0.837, while for the probabilistic model a top 3 accuracy of 0.8791 could be achieved. While the RMSE for the non-probabilistic model is equal for the training and the test set, the top 3 accuracy for the probabilistic model drastically differ from each other. On the



(a) M_1: Final model on the improved images, patch size: 32×32, metric: MSE



(b) Probabilistic M_1: Final model on the improved images, patch size: 32×32, metric: top 3 accuracy

Figure 5.4.: Plots for the respective metric over the epochs trained for the final NNs when using the improved images: The non probabilistic model (Figure 5.4a) shows no significant tendency to overfit the data while for the probabilistic model the network starts to overfit towards the training set after ≈ 80 epochs. As previously described the model is selected based on the best evaluation of the test set.

test set a top 3 accuracy of 0.99926 could be achieved while on the training set only 0.8085 was reached. The respective evolution of the RMSE (Figure 5.4a) and the top 3 accuracy (Figure 5.4b) is shown in Figure 5.4.
When sampling for every object 100 random patches and predicting the FCS for these never seen patches, the mean prediction of the non-probabilistic network per object is still around a seemingly fixed value, 2.5. But because it is not constant in comparison to the old patches, their is a small correlation between the surface structure and the FCS. The same applies for the probabilistic model, here the 2. class representing the area of a FCS between 1.5 and 2.5 is predicted the most.
Nonetheless, from the current point of view, the problem might be rooted in too much noise in the data. While the surface could be enhanced with the new patches, the main root for noise in the patches is the caption on objects like the "Gelatin box" or the "Tuna fish can". In order to reduce the noise for such objects, they need to be removed from the data set, resulting in a too small data set.

5.5 Final Prediction

As the new patches do not provide an image segmentation mask, for every new patch a random image segmentation mask of the corresponding object is selected from the YCB Object set. With this image segmentation mask the mass is estimated by using model M_2 and making use of the lowest variance as previously explained.
In order to generate the data, the previously used split into test and training set for training the neural network was used once again to predict the P_{DC} for the test set. In Table 5.1 an overview of the objects which were used as the test set is given. The FCS and mass prediction are multiplied in order to generate the final prediction of the P_{DC}.
As seen in Table 5.1, the final prediction is not perfectly on point. The actual problem here is the non-variational FCS. There is a small tendency towards the correct value but not a significant change across the tested objects. Thus, one has to remember that the predictions were made for unknown objects on completely unseen new sampled random patches.
Another major problem which occurs by selecting a random set of test objects is the inadmissible mass of the wine glass. The root of the problem are the provided image segmentation masks, where only a few pixels were recognized as the actual objects. Hence, the total area is less than the actual area and the width as well as the height of the wineglass could not be correctly determined.
In the very beginning of this Chapter it was stated that the prediction of grasp forces is possible. After putting the results in perspective an evaluation on the real robotic hand was done.
The final predicted P_{DC} is then set on the robotic hand as the desired P_{DC}. The split of the total sum is done according to Section 3.0.1. For each object 10 trails were executed and the success rate is shown in Table 5.1. A trial is counted as successful, if the hand does not lose contact with the object within two seconds. In this time the slip controller should be possible to stabilize the grasp.

Table 5.1.: Prediction for testing on the real robot: A trail is counted as successful, if the hand does not lose contact within two seconds. Hence, in this time the slip controller should be possible to stabilize the grasp.

Object	Cont. FCS Disc. FCS	True FCS	Pred. Mass	True Mass	Pred. P_{DC}	True P_{DC}	Success %
Knife	2.45451 1	2.073	20.00	31.0	49.09 20.00	64.263	60 10
Strawberry	2.67979 2	3.8687	8.418	18.0	22.558 16.836	64.263	30 10
Wine glass	2.46344 2	3.196	-17.78	133.0	327.638 266	425.035	90 50
Lemon	2.778 2	3.927	55.991	29.0	155.543 111.982	113.889	70 50

As seen in the Table 5.1 the success on the test set is moderate. As both predictions, by the non-probabilistic and probabilistic model, have a relative high error rate, this moderate result could be expected. To use such predictions in a real robotic system it is necessary to incorporate a form of safety margin as already proposed in [5]. With the help of such a safety margin the, most of the time to low predicted, P_{DC} could be bumped up to the potentially correct value. As in Section 4.3.2 the third method was chosen to convolute the P_{DC} into a single value, the P_{DC} to predict is some form of a lower bound. Therefore, if a safety margin is applied to the predicted P_{DC}, it is not likely to enter the area, in which the object may get deformed.

6 Outlook

As this thesis is one of the first approaches in this field, multiple things were simplified. Regarding the grasp itself, it was assumed that the force on the object is always normal onto the object. Such an assumption is not feasible to hold in a non-synthetic environment. Here the measured P_{DC} values might be split along the tangential axis as well as the normal axis of the surface. As only the normal part is the main interest for friction, it must be taken care of how much the normal part accounts into the measured P_{DC}.
Besides that, a more advanced friction model could be used, in order to allow a preciser prediction. With such a friction model the physics can be modeled in a more sophisticated way, resulting in a more complex model, which can be learned, representing the reality better.
Another problem, which can definitely be solved in the future, is the rather small selection of objects. In the YCB Object set a total of 104 objects are provided, but only 24 objects ($\approx 1/4$) were used for this thesis. But using all objects, might bias the models, because sub object sets, like ten marbles (counting as ten single objects), predominate objects, which only occur one time. Therefore only one object of such a sub object set should be used. In addition to that the already mentioned problems (Figure A.1) still occur for some objects, which might withdraw them from the data set.
To improve the FCS prediction new photos were taken in a different lighting environment. As a result, the predicted FCS were improved from constants to moderate adaptive values for the FCS. Currently only ten to fifteen photos were taken per object, thus, the amount of the new photos can be increased in order to extract more patches, resulting in a bigger data set. When the mass is still assumed to be known, the resulting accuracy of the predicted P_{DC} should increase.
An extension to the fixed patch size could be done by using the concept of spatial pyramid pooling introduced by [29]. With this new form of pooling different patch sizes can be used for the same network. Such a new form of input is very interesting, because using different patch sizes, does not restrict the robot to have the same distance to the surface of different objects. One thing to keep in mind with varying patch sizes, is the difference in scaling. From the current point of view, it is important that the patches still cover the same physical area. If the robot can extract the same physical area across different distance with the same camera, the patches will obviously vary in their size depending on the distance to the object. Spatial pyramid pooling would help here by allowing to feed these different patch sizes into the network.
Regarding the segmentation into different bins for encoding the one-hot vector, a more sophisticated approach would be to explicitly cluster the data points available with an off-the shelve algorithm like expectation-maximization and construct based on the clusters the one-hot vector. Thus similar object surface materials may be identified as a cluster, for example cardboard, painted wood or ceramic.
Like previously discussed, relaxing the assumption of the mass known, increases the error, considering the prediction of the mass is not error-free. As a first step the feature extraction could be improved in order to capture more complex features. With the help of more complex features the trained model should generalize better.
A different direction would be, not to use the image segmentation mask at all, removing the need of it, and to calculate the edges of the object based on the photo. From this point the edges can be used to roughly estimate the volume of the object. With the help of the known volume it should be possible to decrease the RMSE already. A simplified approach could be, instead of detecting the edges, searching for basic shapes as a boundary with their scaling in the image. For these shapes the volume is already known and does not need some complicated calculation based on the detected edges. But at least one problem still remains for the two approaches, the density of the object is still unknown. One solution for this rather big problem would be to have some form of database, were the material can be looked up. Having such a database could improve the prediction of the FCS as well, because the FCS could be stored together with the density for different materials. Accordingly, only the material besides the shape of the seen objects needs to be predicted. Nonetheless, from the current point of view storing every possible material does not seem to be the desired solution.
Continuing the separate ideas of combining both predictions into one material and the previously discussed clustering of the FCS, a discrete approach across both dimensions, the FCS as well as the mass, could be tested as well. As a result, each cluster has a fixed FCS as well as a fixed mass. Now the goal is to predict the corresponding class from the whole photo. This approach is similar to cluster only based on the P_{DC}. But with the extra given split (Chapter 3) between mass and FCS, one could assume that the resulting clustering suits the data better. Following this path is open for discussion.
Last, another application of the formulas is given, as the described split is not fixed to predict the P_{DC}. If the variables in Equation 3.3 are reordered, the robot might look shortly at the object, estimate the FCS, then lift the object and adjust the pressure until the grasp is stable. Now with the help of the known P_{DC}, the mass of the object could be estimated by

$$m = \frac{P_{\mathrm{DC}}}{FCS}.$$

In the same manner the FCS could be estimated, when the mass is predicted in advance.

As a conclusion, the proposed methods showed the existence of a correlation between the photo and the grasp force. Hence, it is feasible to predict the grasp forces, but as outlined before, there is room for improvement at multiple points in the suggested method.

Bibliography

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[2] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[3] N. Wettels, V. J. Santos, R. S. Johansson, and G. E. Loeb, "Biomimetic tactile sensor array," *Advanced Robotics*, vol. 22, no. 8, pp. 829–849, 2008-01.

[4] F. Veiga, H. Van Hoof, J. Peters, and T. Hermans, "Stabilizing novel objects by learning to predict tactile slip," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 5065–5072, IEEE, 2015.

[5] Z. Su, K. Hausman, Y. Chebotar, A. Molchanov, G. E. Loeb, G. S. Sukhatme, and S. Schaal, "Force estimation and slip detection for grip control using a biomimetic tactile sensor," in *Proceedings of the International Conference on Humanoid Robots, Seoul, Korea*, pp. 3–5, 2015.

[6] G. De Maria, P. Falco, C. Natale, and S. Pirozzi, "Integrated force/tactile sensing: The enabling technology for slipping detection and avoidance," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 3883–3889, IEEE, 2015.

[7] T. Maeno, S. Hiromitsu, and T. Kawai, "Control of grasping force by detecting stick/slip distribution at the curved surface of an elastic finger," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 4, pp. 3895–3900, IEEE, 2000.

[8] N. Wettels, L. M. Smith, V. J. Santos, and G. E. Loeb, "Deformable skin design to enhance response of a biomimetic tactile sensor," in *Biomedical Robotics and Biomechatronics, 2008. BioRob 2008. 2nd IEEE RAS & EMBS International Conference on*, pp. 132–137, IEEE, 2008.

[9] K. Nakamura and H. Shinoda, "A tactile sensor instantaneously evaluating friction coefficients," in *Transducers' 01 Eurosensors XV*, pp. 1402–1405, Springer, 2001.

[10] S. Tasdemir, A. Urkmez, and S. Inal, "Determination of body measurements on the holstein cows using digital image analysis and estimation of live weight with regression analysis," *Computers and electronics in agriculture*, vol. 76, no. 2, pp. 189–197, 2011.

[11] K. Kollis, C. S. Phang, T. M. Banhazi, and S. J. Searle, "Weight estimation using image analysis and statistical modelling: a preliminary study," *Applied engineering in agriculture*, vol. 23, no. 1, pp. 91–96, 2007.

[12] C. P. Schofield, C. M. Wathes, and A. R. Frost, "Integrated management systems for pigs–increasing production efficiency and welfare.," *Animal Production in Australia*, vol. 24, pp. 197–200, 2002.

[13] C. Chaithanya and S. Priya, "Object weight estimation from 2d images," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 17, 2015.

[14] F. Reuleaux and E. S. Ferguson, *Kinematics of machinery: outlines of a theory of machines*. London, Macmillan, 1876.

[15] A. Bicchi, "On the closure properties of robotic grasping," *The International Journal of Robotics Research*, vol. 14, no. 4, pp. 319–334, 1995-08-01.

[16] P. Somoff, "Über gebiete von schraubengeschwindigkeiten eines starren körpers bei verschiedener zahl von stützflächen.," *Zeitschrift für Mathematik und Physik*, vol. 45, pp. 245–266, 1900.

[17] D. Gross, W. Hauger, J. Schröder, and W. A. Wall, *Technische Mechanik 1*. Springer-Lehrbuch, Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-36268-2.

[18] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[19] J. Fishel, G. Lin, and G. Loeb, “BioTac® product manual,” *SynTouch LLC, February*, 2013.

[20] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[21] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[22] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv:1412.6980 [cs]*, 2014-12-22.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. ICML*, vol. 30, 2013.

[25] Y. Freund and R. E. Schapire, “A desicion-theoretic generalization of on-line learning and an application to boosting,” in *European conference on computational learning theory*, pp. 23–37, Springer, 1995.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems 2, NIPS 1989*, pp. 396–404, Morgan Kaufmann Publishers, 1990.

[27] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, “Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols,” *arXiv:1502.03143 [cs]*, 2015-02-10.

[28] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, “Yale-CMU-berkeley dataset for robotic manipulation research,” *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017-03-01.

[29] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *arXiv:1406.4729 [cs]*, vol. 8691, pp. 346–361, 2014.

A Appendix

A.1 Formulas

From [21], given a marginal Gaussian distribution for $\boldsymbol{x}$ (Equation A.1) and a conditional Gaussian distribution for $\boldsymbol{y}$ given $\boldsymbol{x}$ (Equation A.2) in the form

$$\begin{aligned} p(\boldsymbol{x}) &= \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}\right) && \text{(A.1)} \\ p(\boldsymbol{y} \mid \boldsymbol{x}) &= \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \mathbf{L}^{-1}\right) && \text{(A.2)} \end{aligned}$$

the marginal distribution of $\boldsymbol{y}$ is given by

$$p(\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}\right) \tag{A.3}$$

From [21], suppose $\boldsymbol{x}$ is a D-dimensional vector with Gaussian distribution $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$. Now $\boldsymbol{x}$ is split into two parts $\boldsymbol{x}_a \in \mathcal{R}^M$ and $\boldsymbol{x}_b \in \mathcal{R}^{D-M}$

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{pmatrix}.$$

The mean vector $\boldsymbol{\mu}$

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}.$$

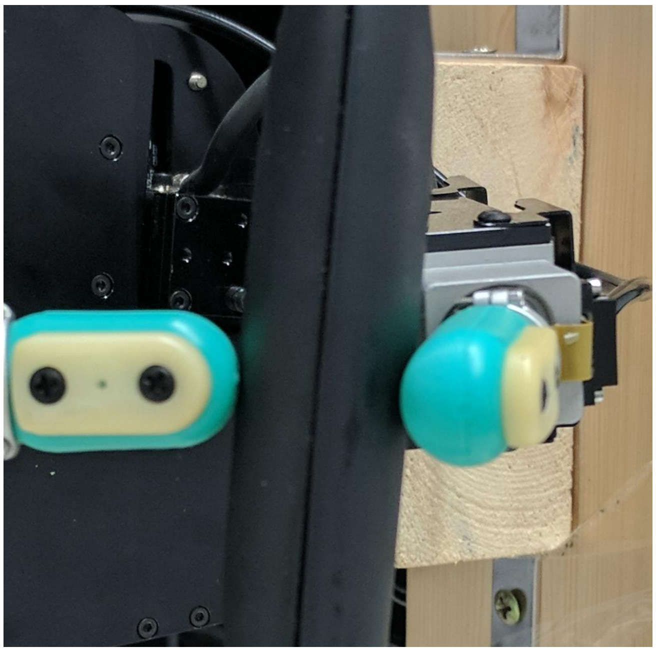
as well as the covariance matrix $\boldsymbol{\Sigma}$

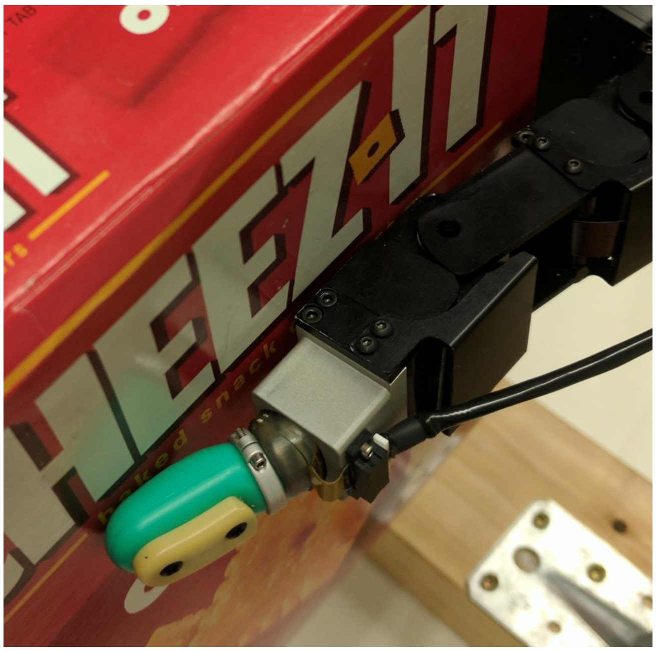
$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}.$$

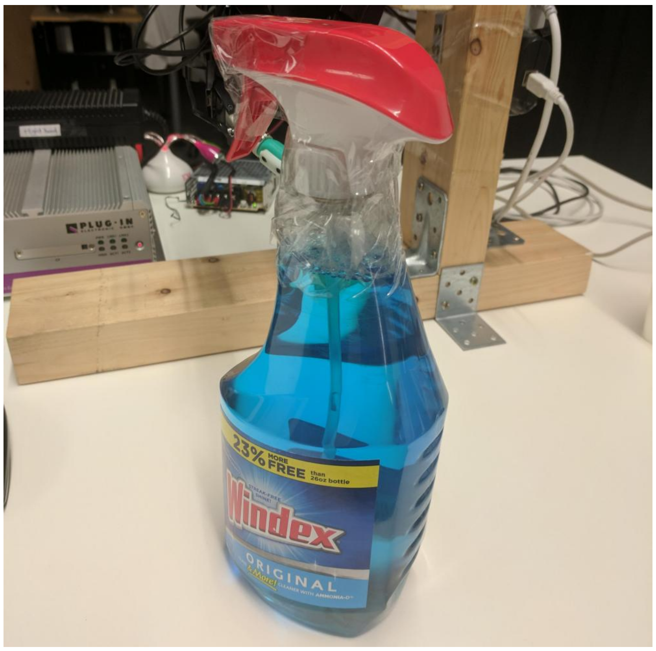
are obtained in a similar way. For such a split the conditional distribution $p(\boldsymbol{x}_a \mid \boldsymbol{x}_b)$ is known to be a Gaussian distribution $\mathcal{N}\left(\boldsymbol{x}_a \mid \boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b}\right)$ with the parameter defined as:

$$\begin{aligned} \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\left(\boldsymbol{x}_b - \boldsymbol{\mu}_b\right) && \text{(A.4)} \\ \boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} && \text{(A.5)} \end{aligned}$$

A.2 Problematic Grasps


(a) When the most front part of the finger tip is pressed no change in static pressure can be measured.


(b) If the object is too big, it might not only touch the finger tip but other parts of the hand as well. Applied pressure through other parts can not be measured.


(c) The object is just too heavy.

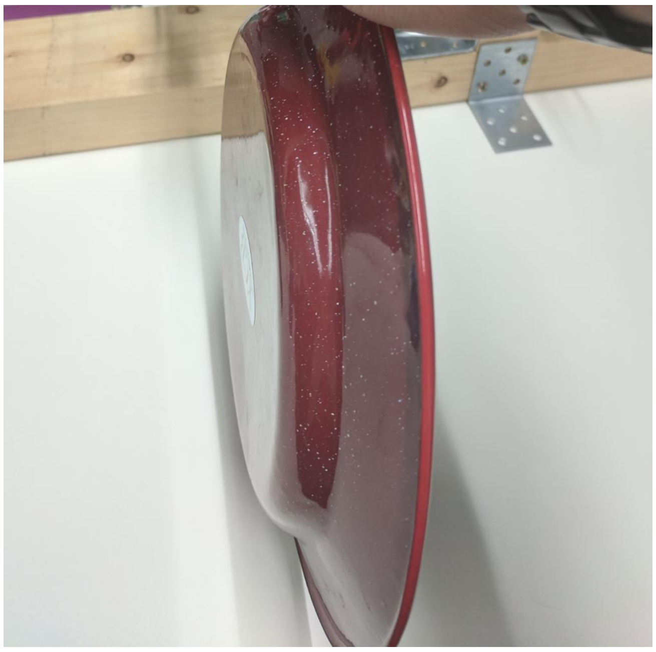

(d) There is no reasonable finger placement onto the object to solely be a force grasp.

Figure A.1.: Problems that might occur when grasping an object. Problem a) and b) could be resolved by grasping the object again. Problem c) and d) discard the objects from the possible object set.

A.3 Object List

Table A.1.: All 24 objects used for training

Object ID/Name	Mass [m]	FCS	P_{DC}	Photo
Chips can	205.0	2.48	507.91	
Cracker box	411.0	1.46	599.34	
Tomato soup can	349.0	2.11	736.16	
Tuna fish can	171.0	2.04	349.53	
Pudding box	187.0	1.63	304.26	

Table A.1.: All 24 objects used for training

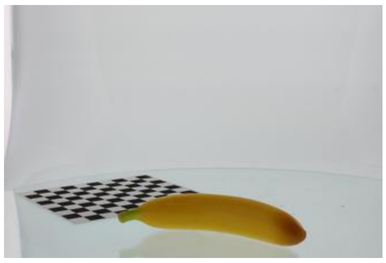
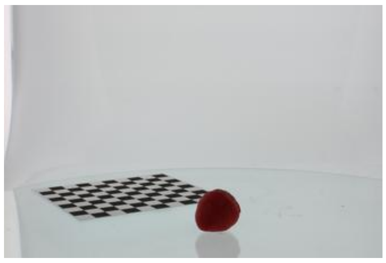
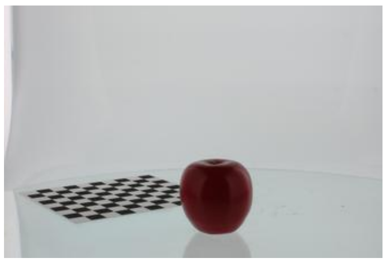
Object ID/Name	Mass [m]	FCS	P_{DC}	Photo
Gelatin box	97.0	1.05	102.20	
Potted meat can	370.0	2.81	1038.97	
Banana	66.0	2.72	179.84	
Strawberry	18.0	3.87	69.64	
Apple	68.0	3.53	240.37	

Table A.1.: All 24 objects used for training

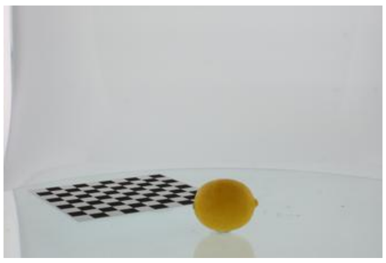
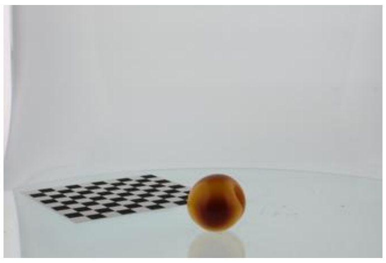
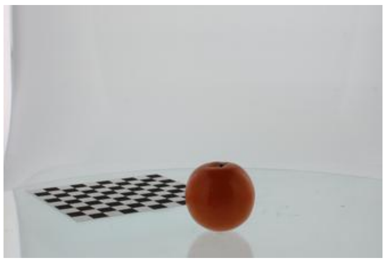
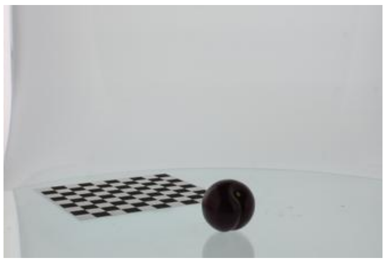
Object ID/Name	Mass [m]	FCS	P_{DC}	Photo
Lemon	29.0	3.93	113.89	
Peach	33.0	2.96	97.82	
Orange	47.0	7.28	342.11	
Plum	25.0	2.82	70.61	
Wine glass	133.0	3.20	425.04	

Table A.1.: All 24 objects used for training

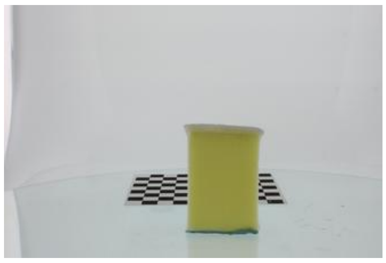
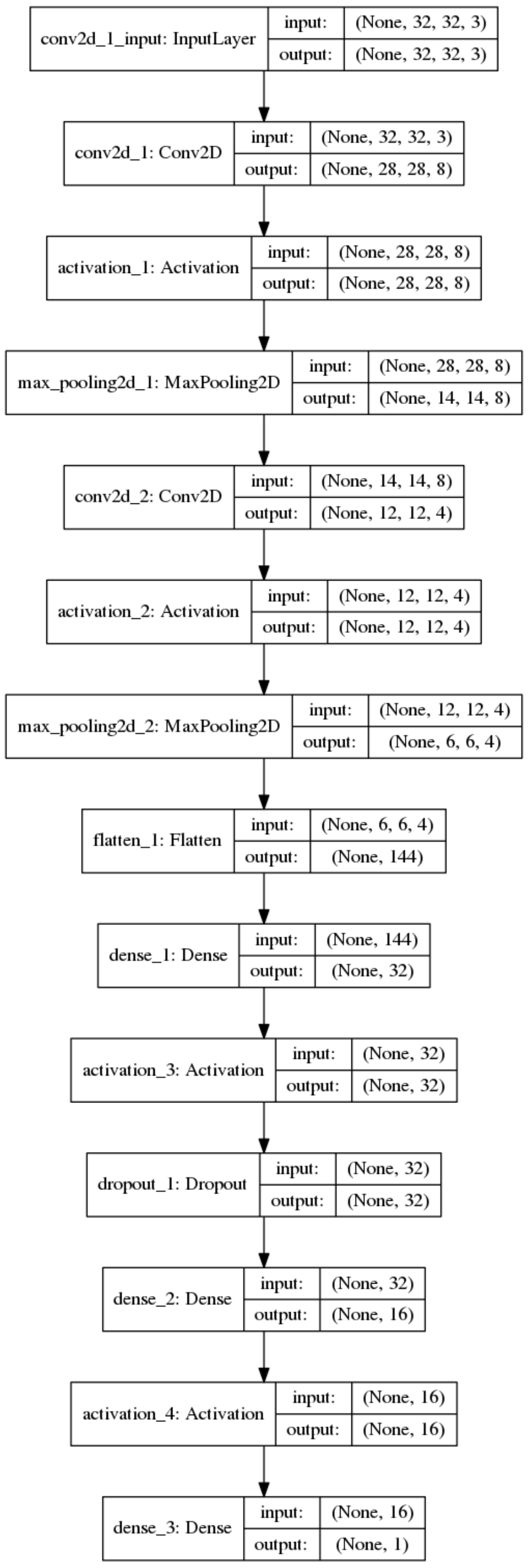
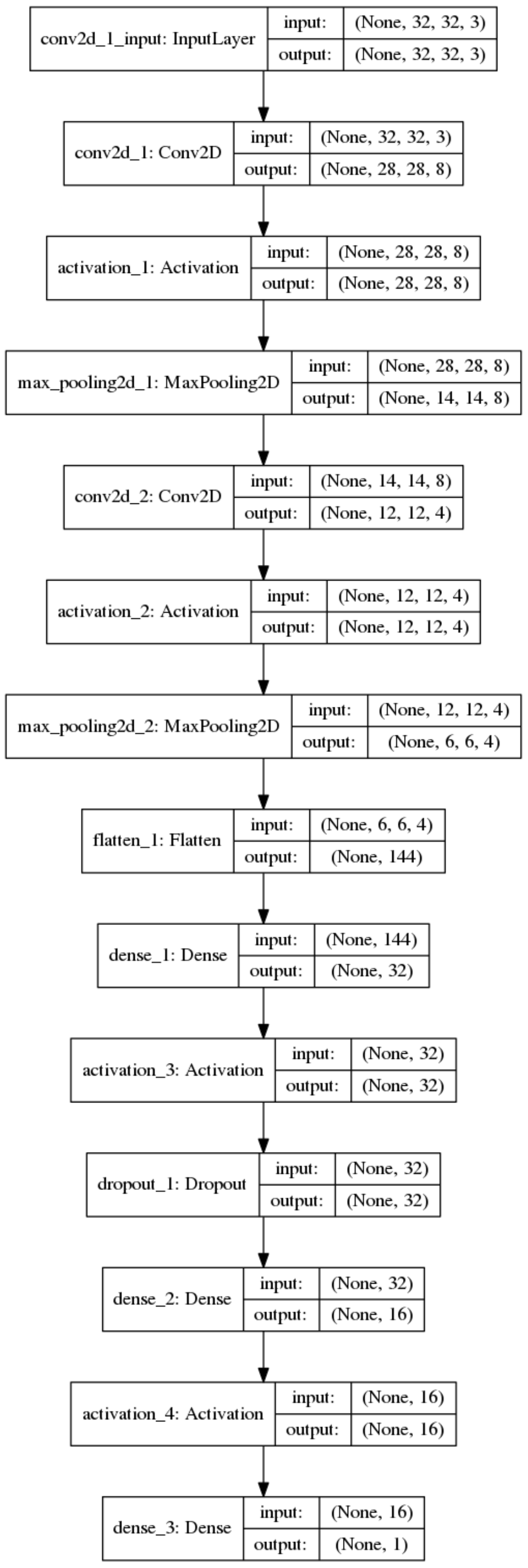
Object ID/Name	Mass [m]	FCS	P_{DC}	Photo
Sponge	6.2	5.88	36.47	
Fork	34.0	1.85	62.85	
Spoon	30.0	2.01	60.44	
Knife	31.0	3.16	97.92	
Spatula	51.5	2.71	139.31	

Table A.1.: All 24 objects used for training

Object ID/Name	Mass [m]	FCS	P_{DC}	Photo
Scissors	82.0	2.70	221.21	
Large marker	15.8	4.07	64.29	
Small marker	8.2	3.94	32.33	
Phillips screwdriver	97.0	2.62	253.96	

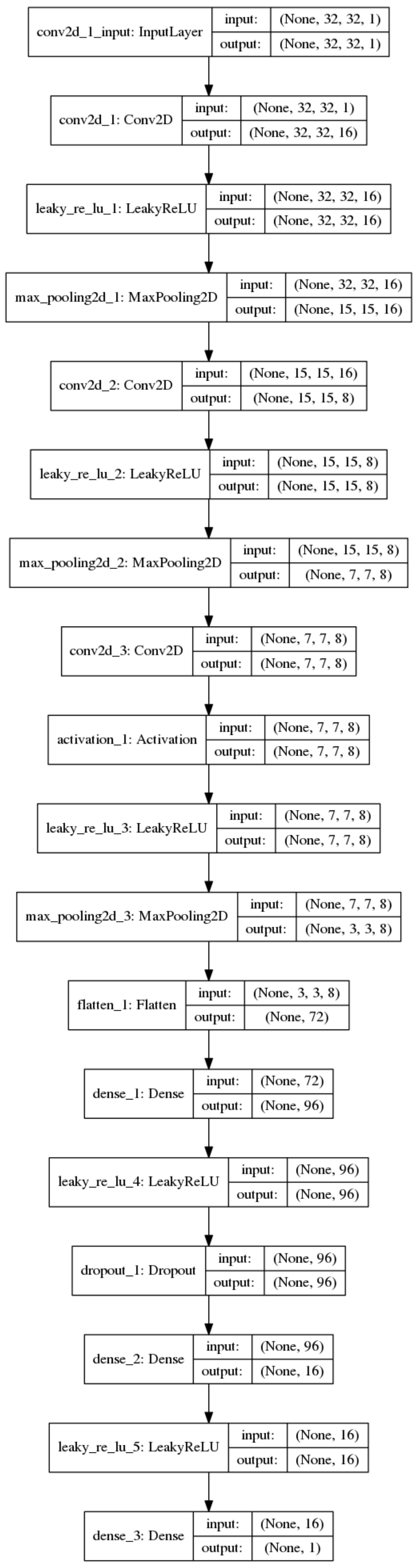

conv2d_1_input: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)
conv2d_1: Conv2D | input: (None, 32, 32, 3) | output: (None, 28, 28, 8)
activation_1: Activation | input: (None, 28, 28, 8) | output: (None, 28, 28, 8)
max_pooling2d_1: MaxPooling2D | input: (None, 28, 28, 8) | output: (None, 14, 14, 8)
conv2d_2: Conv2D | input: (None, 14, 14, 8) | output: (None, 12, 12, 4)
activation_2: Activation | input: (None, 12, 12, 4) | output: (None, 12, 12, 4)
max_pooling2d_2: MaxPooling2D | input: (None, 12, 12, 4) | output: (None, 6, 6, 4)
flatten_1: Flatten | input: (None, 6, 6, 4) | output: (None, 144)
dense_1: Dense | input: (None, 144) | output: (None, 32)
activation_3: Activation | input: (None, 32) | output: (None, 32)
dropout_1: Dropout | input: (None, 32) | output: (None, 32)
dense_2: Dense | input: (None, 32) | output: (None, 16)
activation_4: Activation | input: (None, 16) | output: (None, 16)
dense_3: Dense | input: (None, 16) | output: (None, 1)

(a) M_1: Baseline model, patch size: 32×32

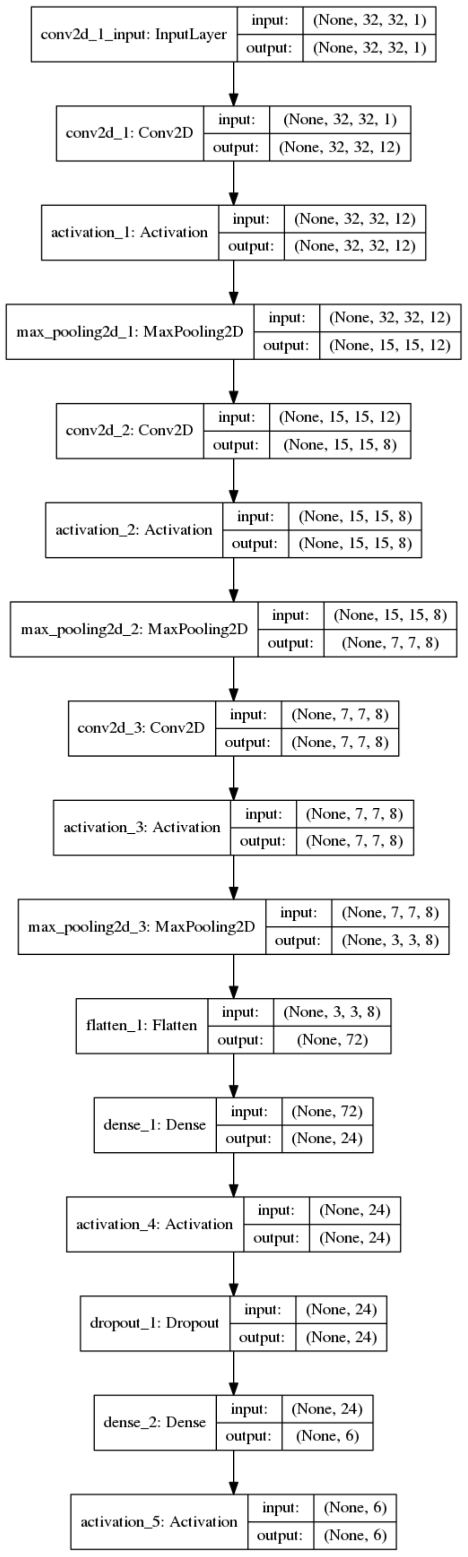

conv2d_1_input: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)
conv2d_1: Conv2D | input: (None, 32, 32, 3) | output: (None, 28, 28, 8)
activation_1: Activation | input: (None, 28, 28, 8) | output: (None, 28, 28, 8)
max_pooling2d_1: MaxPooling2D | input: (None, 28, 28, 8) | output: (None, 14, 14, 8)
conv2d_2: Conv2D | input: (None, 14, 14, 8) | output: (None, 12, 12, 4)
activation_2: Activation | input: (None, 12, 12, 4) | output: (None, 12, 12, 4)
max_pooling2d_2: MaxPooling2D | input: (None, 12, 12, 4) | output: (None, 6, 6, 4)
flatten_1: Flatten | input: (None, 6, 6, 4) | output: (None, 144)
dense_1: Dense | input: (None, 144) | output: (None, 32)
activation_3: Activation | input: (None, 32) | output: (None, 32)
dropout_1: Dropout | input: (None, 32) | output: (None, 32)
dense_2: Dense | input: (None, 32) | output: (None, 16)
activation_4: Activation | input: (None, 16) | output: (None, 16)
dense_3: Dense | input: (None, 16) | output: (None, 1)

(b) Probabilistic M_1: Baseline model, patch size: 32×32

Figure A.2.: These Neural Network models serves as the baseline for all following models


conv2d_1_input: InputLayer | input: (None, 32, 32, 1) | output: (None, 32, 32, 1)
conv2d_1: Conv2D | input: (None, 32, 32, 1) | output: (None, 32, 32, 16)
leaky_re_lu_1: LeakyReLU | input: (None, 32, 32, 16) | output: (None, 32, 32, 16)
max_pooling2d_1: MaxPooling2D | input: (None, 32, 32, 16) | output: (None, 15, 15, 16)
conv2d_2: Conv2D | input: (None, 15, 15, 16) | output: (None, 15, 15, 8)
leaky_re_lu_2: LeakyReLU | input: (None, 15, 15, 8) | output: (None, 15, 15, 8)
max_pooling2d_2: MaxPooling2D | input: (None, 15, 15, 8) | output: (None, 7, 7, 8)
conv2d_3: Conv2D | input: (None, 7, 7, 8) | output: (None, 7, 7, 8)
activation_1: Activation | input: (None, 7, 7, 8) | output: (None, 7, 7, 8)
leaky_re_lu_3: LeakyReLU | input: (None, 7, 7, 8) | output: (None, 7, 7, 8)
max_pooling2d_3: MaxPooling2D | input: (None, 7, 7, 8) | output: (None, 3, 3, 8)
flatten_1: Flatten | input: (None, 3, 3, 8) | output: (None, 72)
dense_1: Dense | input: (None, 72) | output: (None, 96)
leaky_re_lu_4: LeakyReLU | input: (None, 96) | output: (None, 96)
dropout_1: Dropout | input: (None, 96) | output: (None, 96)
dense_2: Dense | input: (None, 96) | output: (None, 16)
leaky_re_lu_5: LeakyReLU | input: (None, 16) | output: (None, 16)
dense_3: Dense | input: (None, 16) | output: (None, 1)

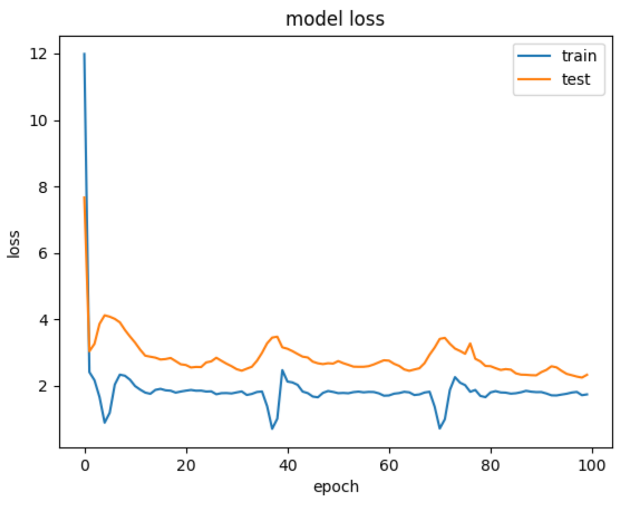
(a) M_1: Improved images model, patch size: 32×32


conv2d_1_input: InputLayer | input: (None, 32, 32, 1) | output: (None, 32, 32, 1)
conv2d_1: Conv2D | input: (None, 32, 32, 1) | output: (None, 32, 32, 12)
activation_1: Activation | input: (None, 32, 32, 12) | output: (None, 32, 32, 12)
max_pooling2d_1: MaxPooling2D | input: (None, 32, 32, 12) | output: (None, 15, 15, 12)
conv2d_2: Conv2D | input: (None, 15, 15, 12) | output: (None, 15, 15, 8)
activation_2: Activation | input: (None, 15, 15, 8) | output: (None, 15, 15, 8)
max_pooling2d_2: MaxPooling2D | input: (None, 15, 15, 8) | output: (None, 7, 7, 8)
conv2d_3: Conv2D | input: (None, 7, 7, 8) | output: (None, 7, 7, 8)
activation_3: Activation | input: (None, 7, 7, 8) | output: (None, 7, 7, 8)
max_pooling2d_3: MaxPooling2D | input: (None, 7, 7, 8) | output: (None, 3, 3, 8)
flatten_1: Flatten | input: (None, 3, 3, 8) | output: (None, 72)
dense_1: Dense | input: (None, 72) | output: (None, 24)
activation_4: Activation | input: (None, 24) | output: (None, 24)
dropout_1: Dropout | input: (None, 24) | output: (None, 24)
dense_2: Dense | input: (None, 24) | output: (None, 6)
activation_5: Activation | input: (None, 6) | output: (None, 6)

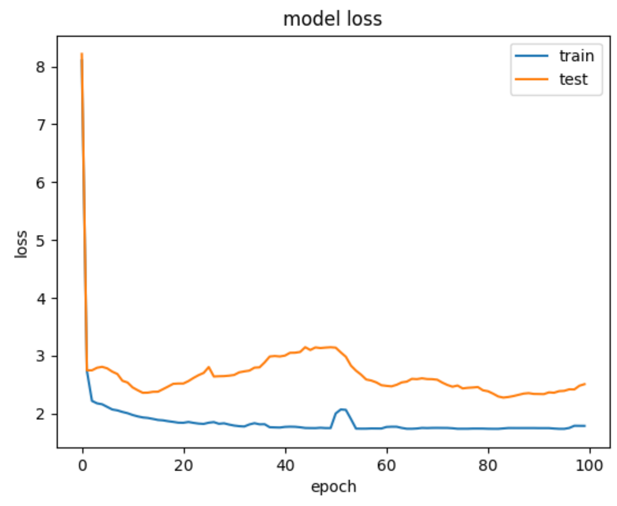
(b) Probabilistic M_1: Improved images model, patch size: 32×32

Figure A.3.: The final NN structure for the improved images

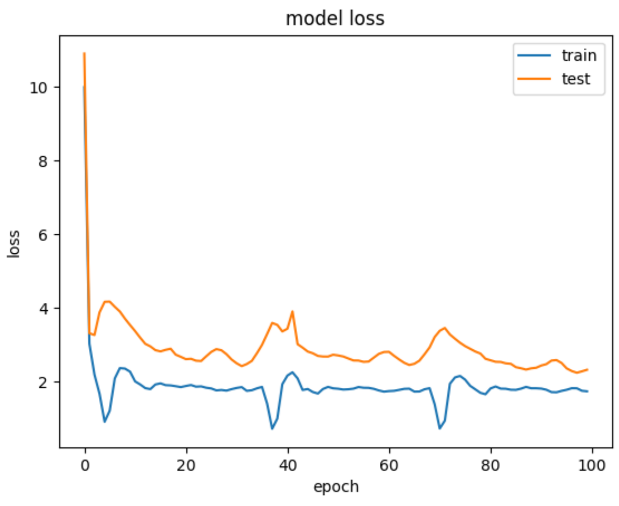
A.5 Neural Network Convergence



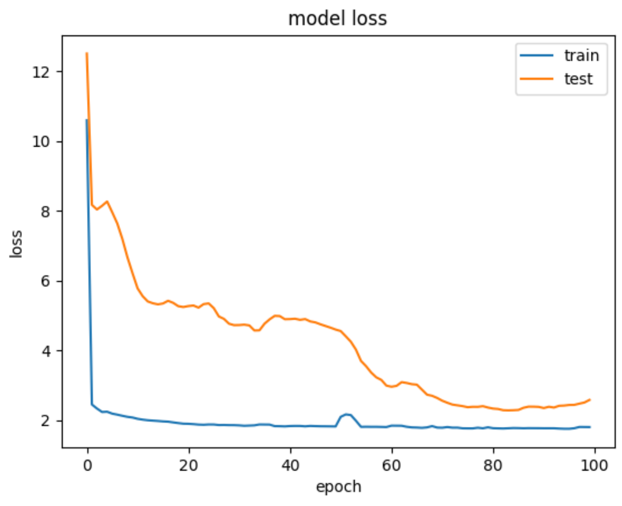
(a) M_1: baseline model, patch size: 32×32



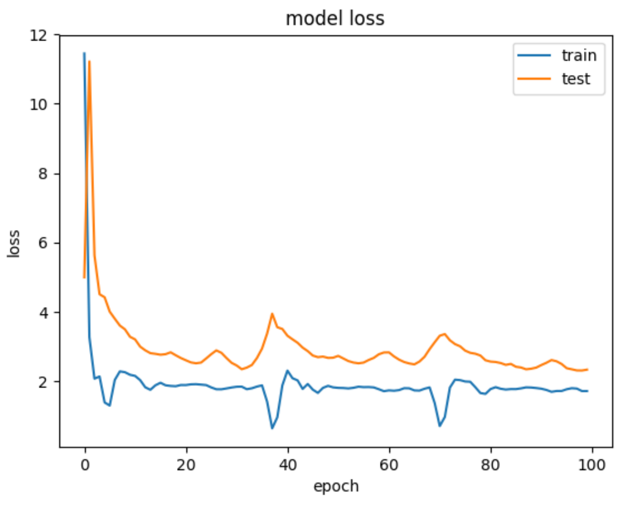
(b) M_1: baseline model, patch size: 64×64



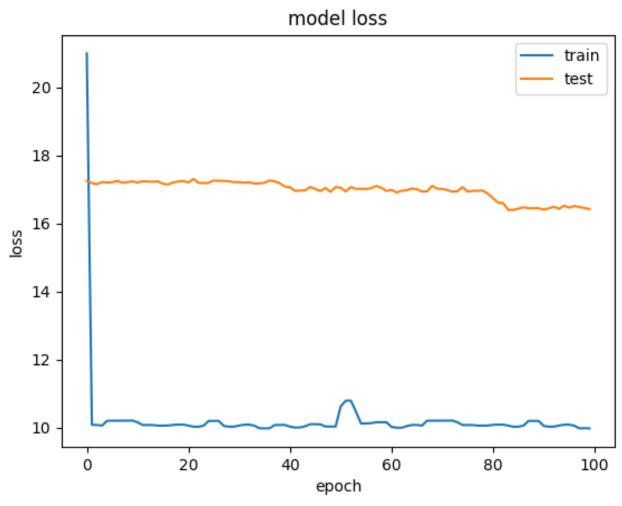
(c) M_1: 2. model, patch size: 32×32



(d) M_1: 2. model, patch size: 64×64

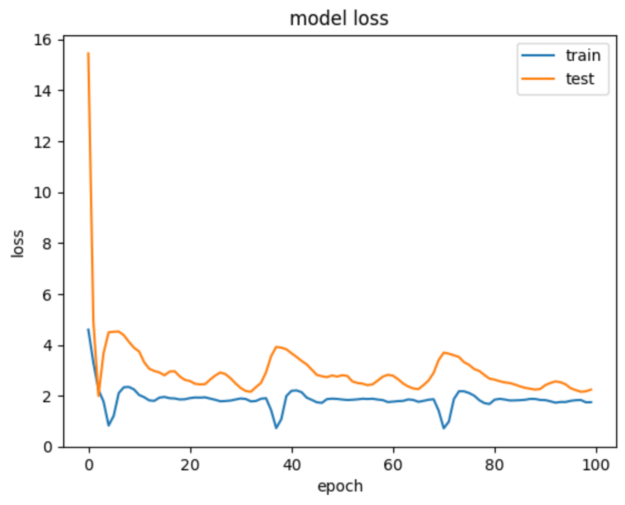


(e) M_1: 3. model, patch size: 32×32

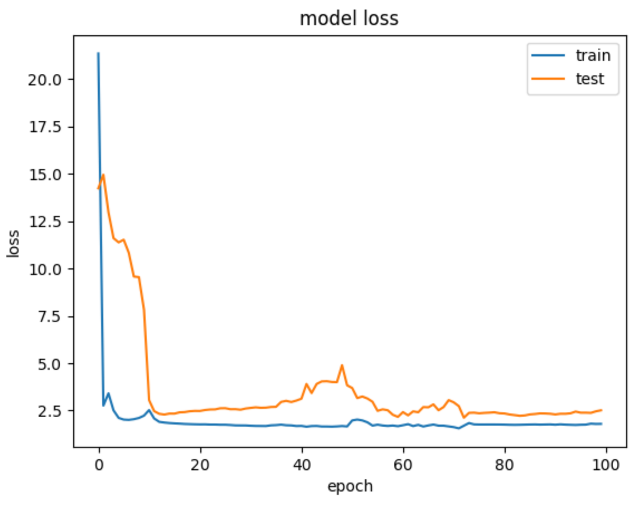


(f) M_1: 3. model, patch size: 64×64

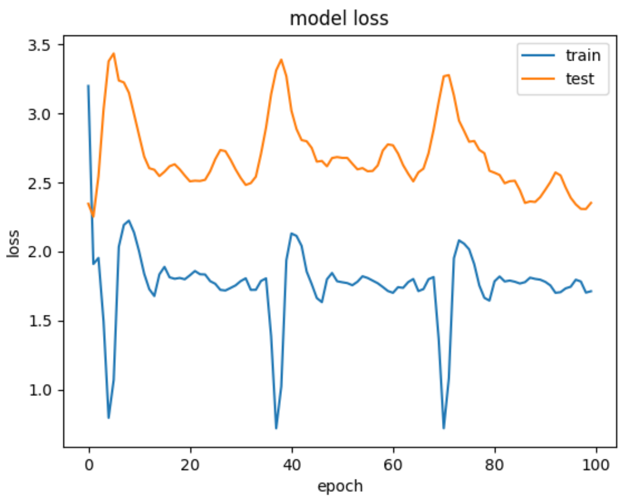
Figure A.4.: Plots for the loss over the epochs trained for all NNs



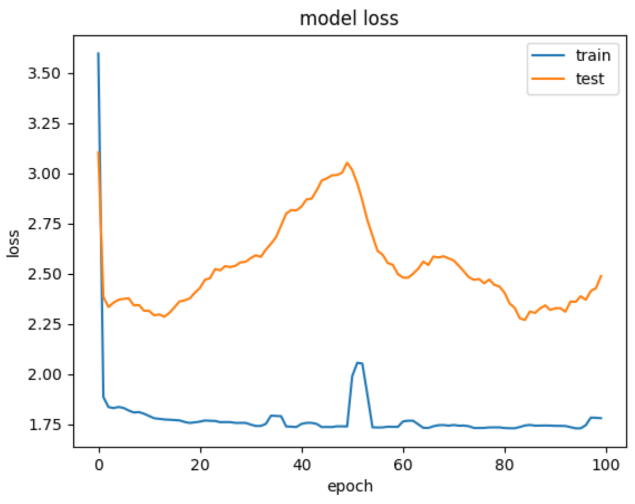
(g) M_1: 4. model, patch size: 32×32



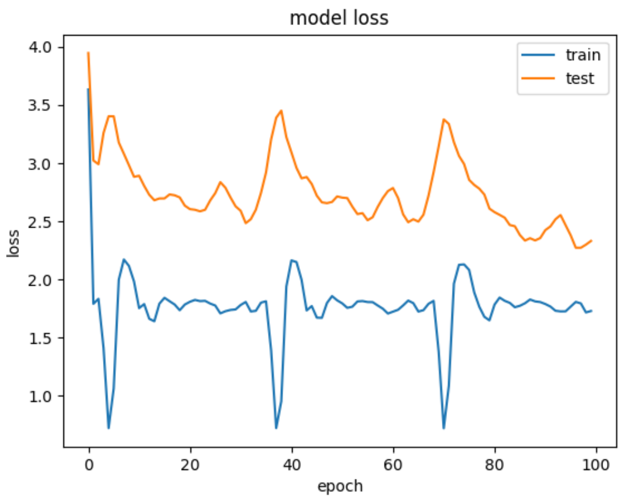
(h) M_1: 4. model, patch size: 64×64



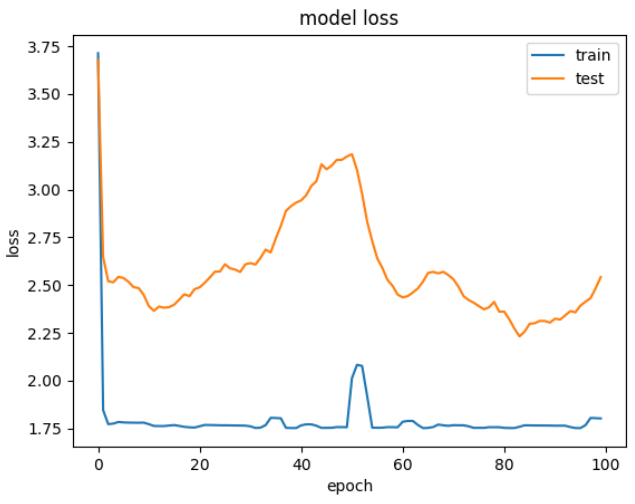
(i) M_1: 5. model, patch size: 32×32



(j) M_1: 5. model, patch size: 64×64



(k) M_1: 6. model, patch size: 32×32



(l) M_1: 6. model, patch size: 64×64

Figure A.4.: Plots for the MSE over the epochs trained for all NNs