University of Southern Denmark

Bachelor thesis

Robot Systems 6th semester - Spring 2019

Neural Networks for Object Classification and Localization of Consumables in a Robotic Picking Application

Emil Vincent Ancker
090697
emanc16@student.sdu.dk

Mathias Emil Slettemark-Nielsen
230295
masle16@student.sdu.dk

Mikkel Larsen
080896
milar16@student.sdu.dk

Supervisor: Anders Glent Buch

Project period: 01.02.2019 - 03.06.2019

Abstract

This bachelor thesis is based on a problem provided by the company Qumec Production Aps, who is developing a robot cell targeted at manipulating consumables. The primary focus of the thesis is classification of consumables and briefly cover the areas of localization and object detection.

Prior to the development of methods for classification, localization and object detection, fundamental background knowledge of neural networks for classification is covered, making it possible to understand the design of neural networks.

Throughout the report a fully connected neural network, and two convolutional neural networks are developed and compared with a traditional computer vision method for classification.

Using a block structure consisting of a convolutional layer with ReLU activation function followed by a max pooling layer, it is discovered that a convolutional neural network consisting of 3 blocks achieves the highest classification accuracy.

Localization techniques is presented and evaluated in the report both for usage as a independent method and in combination with a neural network for object detection, where it is shown that using a method building on background subtraction is superior compared to backprojection.

Comparing the designed combination of localization and classification with a state of the art neural network for object detection showed interesting results. It is shown that the state of the art network achieves the best localization scores, but the developed object detection with a combination of localization and a neural network achieves very fast prediction time and a high classification accuracy. Both object detection methods shows to be superior compared with a traditional computer vision method, template matching.

Preface

The report is written as a 15 ECTS bachelor thesis at the Technical Faculty, University of Southern Denmark. The problem for which the bachelor thesis is based upon is a project under development at the company Qumec Production Aps. The primary work were made at University of Southern Denmark, while the data acquisition was done at Qumec. The project group's supervisor throughout the whole project has been Anders Glent Buch from SDU Robotics.

Prior to the development of the methods described in this thesis, the course *CS231n: Convolutional Neural Networks for Visual Recognition* provided by Stanford University[1] was followed by the project group in the beginning of the project period. This was done to form a foundational background knowledge of the subjects needed for this bachelor thesis.

During the report, "ground truth" will be used as a reference to the correct label of an image. Furthermore, model "weights" and model "parameters" will be used interchangeably. Additionally, when a region is mentioned during this report only rectangular regions are considered.

Summary of Notation

Throughout the report, vectors will be represented by boldface lowercase letters, while matrices will be represented by boldface capital letters.

$\boldsymbol{s}$	vector containing scores calculated by a neural network for an image
$\boldsymbol{\hat{y}}$	vector containing the output of softmax for an image, when given scores as input.
$\boldsymbol{y}$	vector containing ground truth for an image.
$\boldsymbol{W}$	matrix containing weights
$\boldsymbol{x}$	vector representing input
$\boldsymbol{b}$	bias vector
f	used for representing a score function
h	used to represent nonlinear activation function

[1] `http://cs231n.stanford.edu/`

Contents

1 Project Description

A big trend in manufacturing is to make the production lines smarter and free humans of repetitive time consuming tasks. Companies today are moving towards increasingly automated solutions, since there are multiple advantages behind automation. Some advantages of moving towards automation is lower production costs through release of human workforce, which also could lead to increased production rate. Furthermore, the production line could become more adaptive to product changes, resulting in lowering of production line conversion costs.

This bachelor thesis explores an area of automation, which is based upon a robotic picking application targeted at factories handling consumables. The specific area being explored is automation through the usage of computer vision. The company Qumec Production Aps is developing a robot cell, which contains a delta robot targeted at manipulating various consumables on a production line. The robot cell is visualized in figure 1.

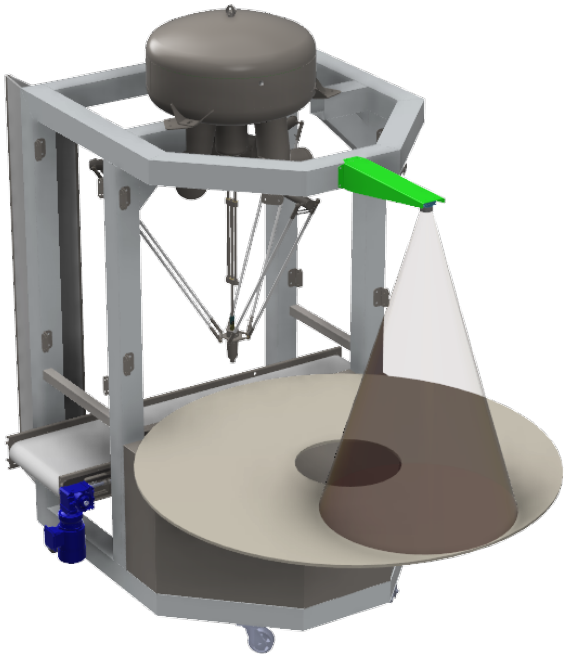


Figure 1: 3D CAD model of the robot cell

The cell contains a rotatable circular table, which can rotate at different angular velocities. Furthermore, a camera is located 140 centimeters above the rotatable table. The delta robot is equipped with a vacuum end effector making it able to grab objects. However, at its current state the robot is only able to grab objects which are located in pre-known positions on the circular table. The robot cell is currently not able to differentiate between

objects, and therefore not able to choose an action based upon the object type.

The company Qumec Production Aps is first and foremost interested in exploring the possibilities of classifying the different consumables, which could make the robot able to act accordingly to the object type. The motivation behind this is to make the robot able to move slower when handling consumables in plastic bags, compared to when manipulating objects in solid containers. Furthermore, this would make the robot able to sort the consumables, which would remove a requirement from the production line of delivering the consumables presorted to the robot cell. Secondly, it is in Qumec Production Aps' interest to have a method for localization of the consumables, so the robot would be able to pick the consumables no matter where they are located on the circular table. Lastly, by combining the classification and localization, the robot cell would be able to pack a mix of different consumables.

The thesis will primarily focus on exploration and development of neural networks for object classification and briefly cover the area of localization and object detection.

1.1 Project Analysis

From the above project description the following problems have been derived.

Collecting data

The robot cell is currently not commercialized. Thus, there are no existing data from the camera. Therefore, collecting data is part of the project. Prior to data collection, it is necessary to investigate acceptable settings of the camera, when capturing images of consumables. Furthermore, the images must be captured in a manner, which is similar to how it is going to be used, to reach a realistic evaluation of the designed methods. Before application of the methods, the data should be preprocessed in a manner, that makes the data easier to interpret.

Classification

To solve the problem of classifying images containing consumables neural networks will be considered and compared with a traditional computer vision technique. Before dwelling into neural networks for classification, the necessary theory of neural networks will need to be covered.

Localization

To make the delta robot able to pick the consumables, different traditional computer vision localization methods must be implemented and evaluated.

Object detection

In order to make the delta robot able to pick different objects and pack a mix of them from a production line, different methods of object detection will be investigated. This can be done in a combination of classification and localization methods, or by using methods directly targeted at object detection.

1.2 Problem Description

From the above mentioned problems the following questions is deduced.

Collecting data

- What are acceptable settings of the camera?
- How should the data be structured?
- How should the data be preprocessed for a neural network?

Classification

- How can the consumables be classified?
- What is a neural network?
- Which library should be used to create neural networks?
- How can the consumables be classified using a neural network?
- Which type of neural network is best at solving the classification problem?

Localization

- How can objects be localized?
- How can regions of interest be found for usage of a classification method?

Object detection

- What is a traditional object detection method?
- How can classification be combined with localization for object detection?
- What is a state of the art method for object detection?
- How does the above three compare?

Throughout the bachelor thesis the above stated problems will be answered.

1.3 Structure of the Report

This thesis starts with an overview of the system and the tools used during the project in section 2, before moving into data collection in section 3. Afterwards, the background

knowledge needed to understand neural networks, when used for image classification, is described in section 4.

Once the initial parts of the project have been covered, the focus will move towards design and evaluation of methods for classification of the consumable in section 5. Here k-Nearest Neighbor will be introduced as a traditional computer vision technique and compared with more modern approaches involving neural networks.

In section 6, two localization techniques, backprojection and background subtraction, and several evaluation metrics for proposal of object locations will be covered.

A combination of classification and localization will be covered in form of object detection in section 7. The section will start with a traditional computer vision technique, Template Matching, before moving into combining a neural network for classification with one of the presented localization techniques. Finally, a state of art neural network targeted at object detection will be evaluated.

Future work, optimization possibilities and problems with the current methods will be discussed in section 8.

Finally, a conclusion of the project work will be covered in section 9.

2 System Overview of the Robot Cell

The robot cell described in section 1, is drawn schematically to create an overview of the problem.

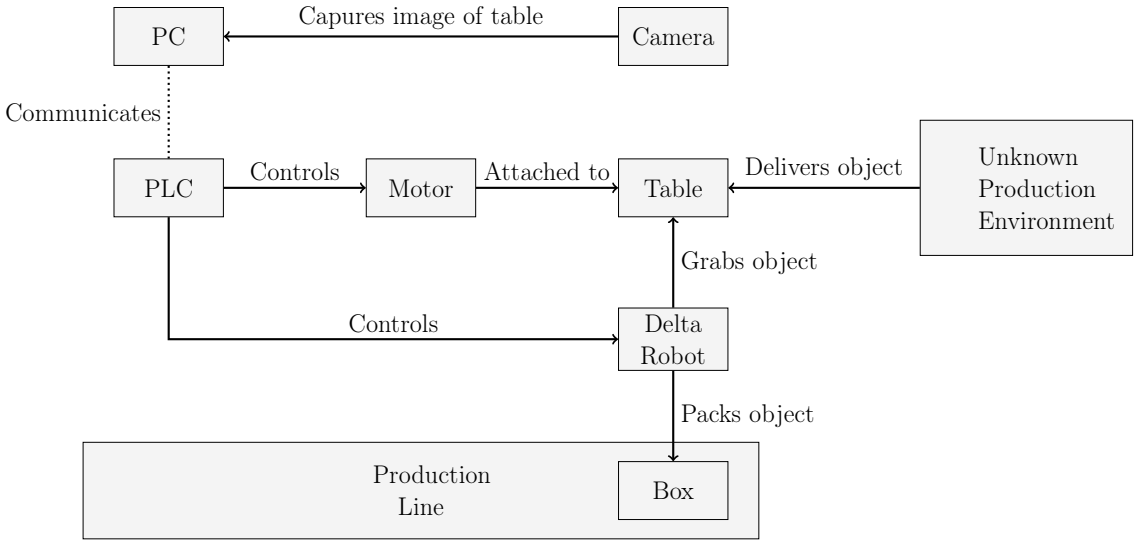


Figure 2: Schematic overview of the robot cell

In the schematic overview in figure 2 it is shown that the camera captures images of the objects on the circular table and sends the images to a PC. The PC is then responsible of processing the images and sending relevant information to the PLC, which controls the robot cell. The focus of this thesis will be primarily on the PC block in the schematic overview.

2.1 Setup of the PC Block

Prior to the design of methods it was necessary to decide, which libraries and tools that were going to be used in the project work.

It was chosen to use the high-level API for neural networks called Keras[2]. Keras uses the machine learning library Tensorflow, which has a library release that supports execution on a graphical processing unit, GPU. Neural networks is highly parallelizable and can benefit from multiple cores, thereby reducing calculation time. Since a GPU has more cores than a CPU, a library with GPU support is preferred. The only programming language compatible with Keras is Python. Both Python and Keras allows for fast prototyping, which is favorable in order to explore the possibilities for classification, localization and object detection.

[2] `https://keras.io/`

For handling packages and virtual environments Anaconda[3] is used on local machines. Later in the project period, the development was moved towards using Google Colaboratory[4], which provides a virtual machine with a more powerfull GPU, than what was available to the project group. This made it possible to handle larger neural networks, but put a restriction on a session runtime of 12 hours before resetting.

The open source computer vision library, OpenCV[5], was used for various computer vision problems in the project work.

The labelling tool Labelbox[6] was used to label the images as ground truth for localization techniques and to train a neural network for object detection.

[3] https://anaconda.org/
[4] https://colab.research.google.com/
[5] https://opencv.org/
[6] https://labelbox.com/

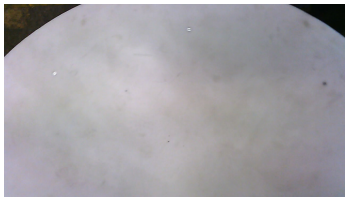
3 Data Collection for Classification and Localization

To solve classification and localization problems of consumables images are gathered. Collecting data is a trivial and time consuming task, since it is necessary to capture images of the objects in different orientations and spatial placements.

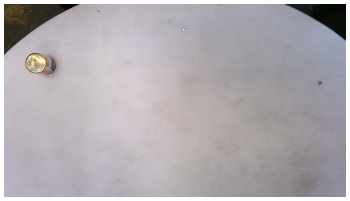
The camera used to gather the images is an ELP-USB13MAF-KV75, ELP (2013). The camera has a maximal resolution of 3840x2880 at a frame rate of 10 frames per second. At a resolution of 1280x720 the camera has a frame rate of 30 frames per second. The latter was chosen as a compromise between image size, quality and frame rate. Furthermore, the phenomenon of motion blur was more significant at the maximal resolution.

First a small dataset, consisting of 505 images in total distributed among 3 classes, was used to solve the problems of classification and localization. However, the dataset was too small to differentiate the methods, thus the dataset was extended. The structure of the small dataset is illustrated in appendix B. The second and larger dataset, consisting of 2347 images distributed among 8 different classes, is used to train the different methods for classification and localization. Example images of the different classes are shown in figure 3.

(a) Arm



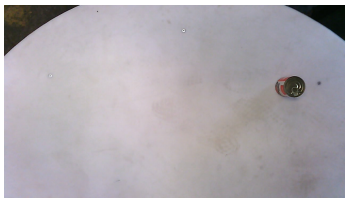
(b) Table



(c) Cat food beef

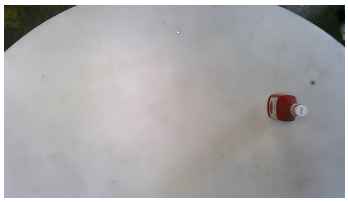
(d) Carrot

(e) Bun



(f) Cat food salmon

(g) Potato



(h) Ketchup

Figure 3: Example images of all classes present in the dataset

The six classes in figure 3c to 3h are different consumables, all relevant for Qumec Production's customers. These classes consists of objects in plastic bags and hard containers. The class in figure 3b is used when there are no objects on the table, and the class in figure 3a is used to recognize arms for safety reasons.

Solving image classification and localization problems normally consists of a training phase, an optimization phase and an evaluation phase. When the three phases are run on the whole dataset, the methods will be good at recognizing exactly what is in the dataset. In reality the visual representation of the classes will never be exactly like the dataset. To obtain the best results of different representations the dataset is normally split into a training set, validation set and a test set to differentiate training, optimization and evaluation. However, the size of the collected dataset was deemed too small. Since the distribution between training and evaluation sets is chosen to be 4:1, as shown in figure 4, the addition of a test set would mean that both the validation and test sets would become

so small, that there is a risk of not describing the populations properly. Therefore, it was chosen to use the validation set as both performance optimization and evaluation.

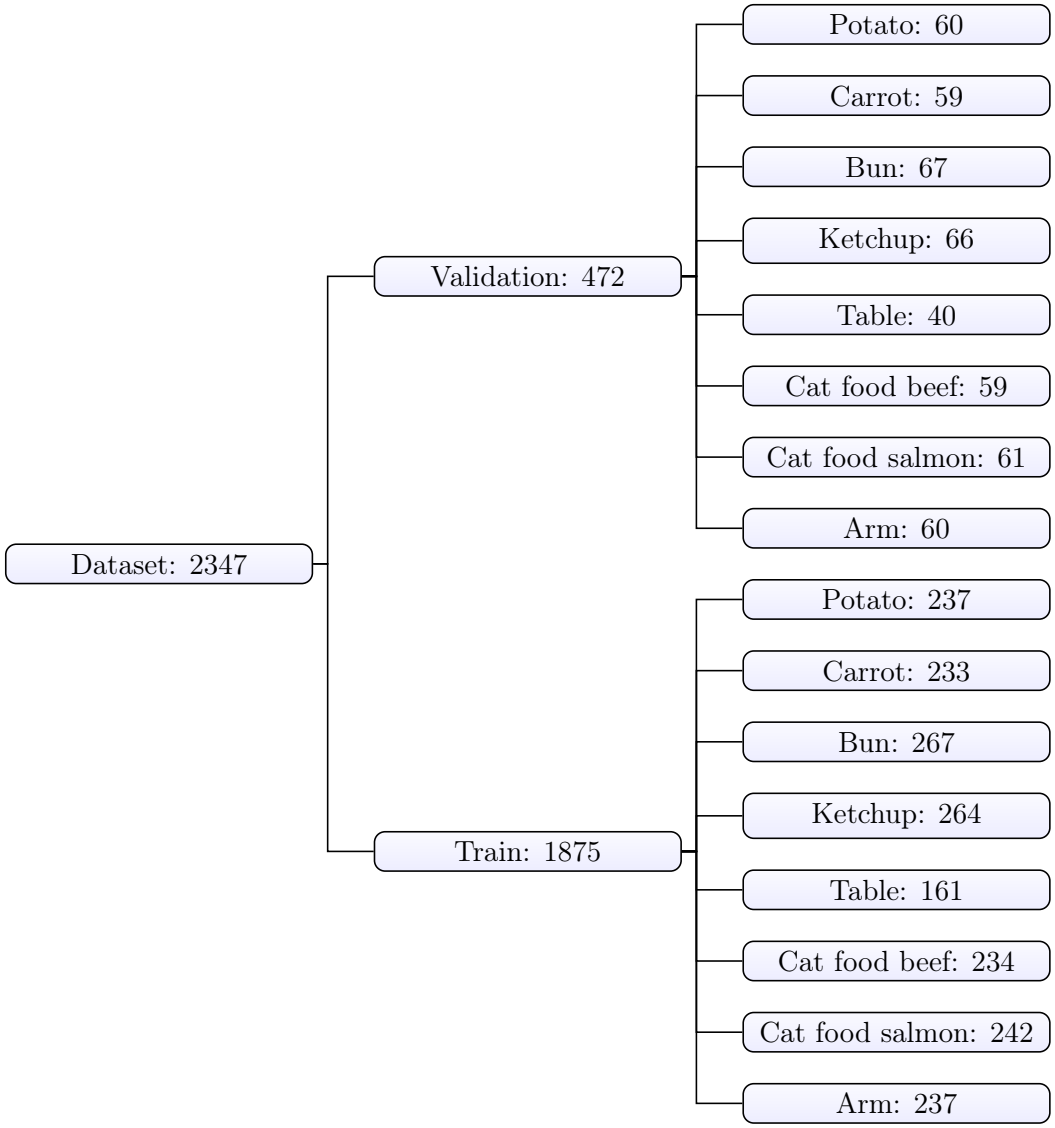


Figure 4: Structure of the large dataset with 8 classes

4 Fundamentals of Neural Networks for Classification

Neural networks is a subject in the domain of artificial intelligence, which have shown to be useful in many applications. In this bachelor thesis the neural networks will be used for computer vision tasks involving classifying and detecting objects in images.

Neural networks is an attempt of recreating the functionality of a biological brain, by creating a structure with a lot of interconnected cells, which makes it possible to train the network into recognizing specific objects. An often drawn analogy to neural networks is that it consists of interconnected neurons. A simplified mathematical structure for one of these neurons is illustrated in figure 5.

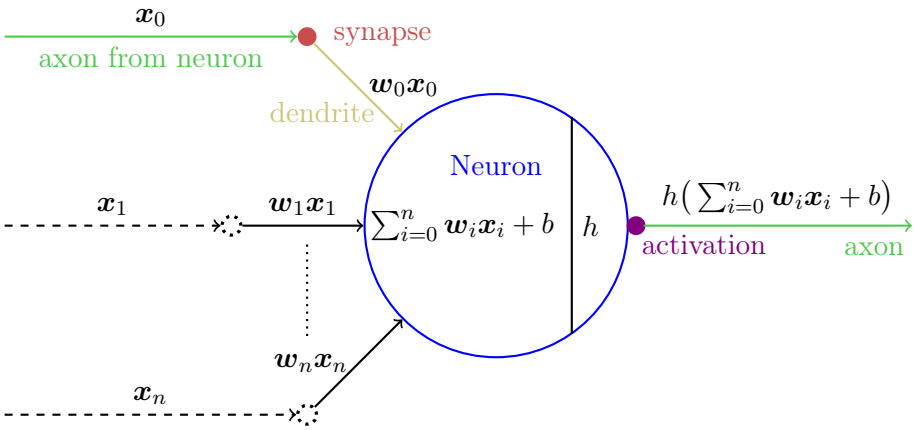


Figure 5: Visualization of a simplified mathematical model of a neuron.

The $\boldsymbol{x}$ variables are used to indicate the outputs coming from previous neurons. $\boldsymbol{w}$ is the weighting introducing from the synapses. The variable b is a bias that is added in each neuron, after summing all inputs. Lastly, the function h is an activation function, which determines the output of the neuron based upon the processed inputs.

The human brain consists of a vast amount of neurons, which acts as computational units. When multiple neurons are connected it is called a neural network. The information flows through the neural network in the following way:

1. The neuron receive inputs from synapses through its dendrites.
2. The information received through all the neuron's dendrites is processed.
3. Based on the processed information the neuron sends an activation through its axon.
4. The axon branches out into multiple synapses which other neurons can connect to through their dendrites.

The idea behind the simplified mathematical model of a neuron in figure 5, is that the

neural network can be trained through learning how to set the values of the weightings in the synapses, $\boldsymbol{w}_i$, to achieve a desired functionality.

Once a neural network has been designed and set up it can basically be seen as a black box consisting of two stages: the learning stage and the prediction stage.

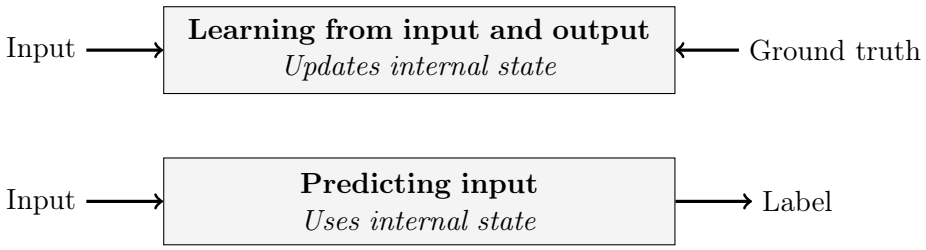


Figure 6: Black box representation of neural networks in training phase and prediction phase.

In this bachelor thesis when a neural network is being trained it will be trained using supervised learning, which implies that the networks learns from examples. These examples will be in form of inputs which are images and outputs which will be labels representing the object classes. As illustrated in figure 6 the network will, during the learning stage, be shown both an input image and the ground truth, thus allowing the network to adjust its internal state making it learn from the shown example. When the neural network is operating in the prediction stage it is shown an input image, then it utilizes its learned internal state making it able to predict a label, which hopefully is correct.

This section will go through the theoretical background knowledge necessary to understand the theory behind the design of neural networks for classification in this bachelor thesis.

4.1 Score Function

When classifying images functionality is needed, which can map the input images into a score for each class. A simple way of achieving this functionality is by using a linear classifier:

$$\underset{c\times 1}{\mathrm{f}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})} = \underset{c\times d}{\boldsymbol{W}} \cdot \underset{d\times 1}{\boldsymbol{x}} + \underset{c\times 1}{\boldsymbol{b}} = \underset{c\times 1}{\boldsymbol{s}} \tag{1}$$

Where $\boldsymbol{x}$ is a flattened image with length d. $\boldsymbol{W}$ is a weight matrix with length c categories and width d. $\boldsymbol{b}$ is bias vector with length c, which is a data independent bias for each class. The output, $\boldsymbol{s}$, is a vector with a score for each category, indirectly representing how much each class is present in the input.

A neural network can be created by stacking linear classifiers with the addition of non

linearity between each classifier. By stacking two linear classifiers with an inbetween non-linear function h a two-layer neural network is created:

$$\boldsymbol{s} = f_{NN}(\boldsymbol{x}, \boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{b_1}, \boldsymbol{b_2}) = f(h(f(\boldsymbol{x}, \boldsymbol{W_1}, \boldsymbol{b_1})), \boldsymbol{W_2}, \boldsymbol{b_2}) \tag{2}$$

f is the function defined in equation 1. The introduced nonlinear function, h, is called an activation function. An activation function is needed to make the neural network, of an architecture illustrated in equation 2, able to approximate any function according to the paper Leshno et al. (1993). Without the addition of an activation function the neural network could just be seen as a linear classifier.

4.2 Loss Function

Once a function mapping input to a score is defined the next needed is a loss function. The purpose of the loss function is to measure the performance of a neural network based on the calculated score. The loss function, L_i, is a function of the calculated scores, $\boldsymbol{s}$, and the ground truth, $\boldsymbol{y}$.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(\boldsymbol{s}_i, \boldsymbol{y}_i) \tag{3}$$

In the above equation N is the total amount of data while i denotes the i'th input data.

4.2.1 Categorical Cross Entropy

The loss function considered is categorical cross entropy, CCE. CCE is a loss function that is well suited for problems involving multiple classes. The method CCE builds upon the statistical method cross entropy, which in its core essence is used to quantify the difference between two probability distributions. Therefore, this puts a restriction upon the calculated scores of the neural network, for which each value should be bounded between 0 and 1, indicating the probability of an image being part of a given category. This will be denoted as $\hat{\boldsymbol{y}}$.

Cross entropy is defined in Dunne and Campbell (1997) as:

$$L_i = \boldsymbol{y}_i \cdot \log\left(\frac{\boldsymbol{y}_i}{\hat{\boldsymbol{y}}_i}\right) \tag{4}$$

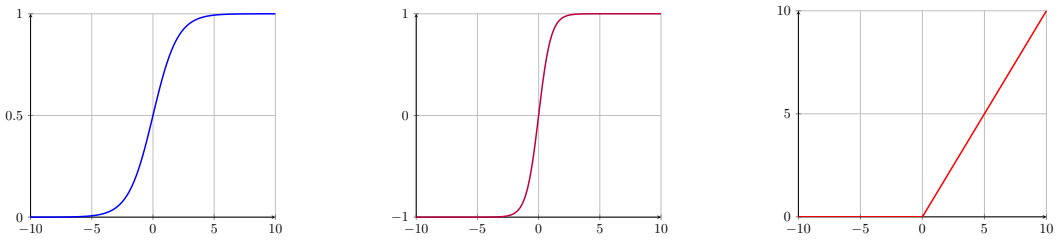
Where $\boldsymbol{y_i}$ is the target probability distribution, $\hat{\boldsymbol{y}}_i$ is the estimate probability distribution. Using the fact that the target probability estimation, in the sense of classification, is 1 for

the ground truth label and 0 for all others, the loss function for CCE can be simplified as:

$$L_i = -\boldsymbol{y}_i \cdot \log\left(\hat{\boldsymbol{y}}_i\right) \tag{5}$$

4.3 Activation Functions

In section 4.1 it was described that introducing a non-linear activation function would make a neural network, with a simple architecture, able to approximate any function. Historically popular activation functions are sigmoid and tanh. Instead of using the sigmoid or tanh activation functions modern neural networks is moving towards using the Rectified Linear Unit, ReLU, activation function as stated in the paper LeCun et al. (2015).



(a) Sigmoid (b) Tanh (c) ReLU

Figure 7: Visualization of the considered activation functions

The sigmoid activation function shown in figure 7a is defined by equation 6.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

A drawback of using the sigmoid activation function is that it forces the gradient towards 0, when the magnitude of the input is near or above 10. This is a problem since the gradient is used during backpropagation explained in section 4.4. Furthermore, the exponential function in equation 6 is expensive to calculate.

The tanh activation function shown in figure 7b is defined by equation 7.

$$t(x) = tanh(x) \tag{7}$$

The tanh activation function solves the problem of zero centered output, seen with the sigmoid function, but it still forces the gradient towards 0, when the magnitude of the input is near or above 10.

The ReLU activation function shown in figure 7c is defined by equation 8.

$$f(x) = max(0, x) \tag{8}$$

ReLU does not saturate in the positive region, which implies that it does not force the gradients towards 0 in the positive region. Besides this ReLU is also very computationally simple and efficient. Furthermore, networks using ReLU converges approximately six times faster than a neural network using the tanh activation function, as stated in the paper Krizhevsky et al. (2012). Some drawbacks of ReLU is that the output is not zero centered, and the gradient is 0 when the input is in the negative region.

ReLU is chosen as the default activation function between the layers, since it is fast to compute and has shown remarkable results in the paper Krizhevsky et al. (2012).

Since it was decided to use categorical cross entropy as a loss function, it puts a restriction upon the calculated scores being a probability distribution. This implies that the activation function applied on the final layer needs to adhere to this requirement. An activation function that falls to mind could be arg max, which sets the highest argument to 1 and the others to 0. This function is not differentiable, therefore it can not be used, the reason behind why it needs to be differentiable will be explained in section 4.4.

Softmax is an activation function that complies with both requirements: the scores needs to be calculated as a probability distribution and it needs to be differentiable. Softmax is defined by:

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{s}_i) = \frac{e^{\boldsymbol{s}_i}}{\sum_{j=1}^{c} e^{\boldsymbol{s}_j}} \tag{9}$$

It is seen that softmax uses the exponential function in the normalization of the output scores, this is used since the exponential function is extraordinary easy to differentiate. Furthermore, it creates a exponential relationship between the scores, making the highest input seem even more relatively high after the application of softmax.

4.4 Backpropagation

The next step, in the process of defining neural networks, is to go through the initial building block of the training phase. When a neural network is training the objective is to minimize the loss. In order to minimize the loss, it is needed to calculate the gradient of the parameters present in the neural network with respect to the loss function. To calculate the gradients of the models parameters with respect to the loss function the technique backpropagation, which builds upon the chain rule and computational graphs,

is used. The computational graph is build from blocks of operations and functions, which are represented by circles in figure 8.

When backpropagation operates, it starts at the end of network and goes backwards through the network as a computational graph, utilizing the chain rule to get the derivatives of the parameters with respect to the loss function. This implies, that all blocks in the computational graphs needs to be differentiable, otherwise the gradients can not be calculated.

An example of backpropagation is shown in figure 8 for a simple two layer neural network, with ReLU activation function after the first layer and softmax activation function after the last layer.

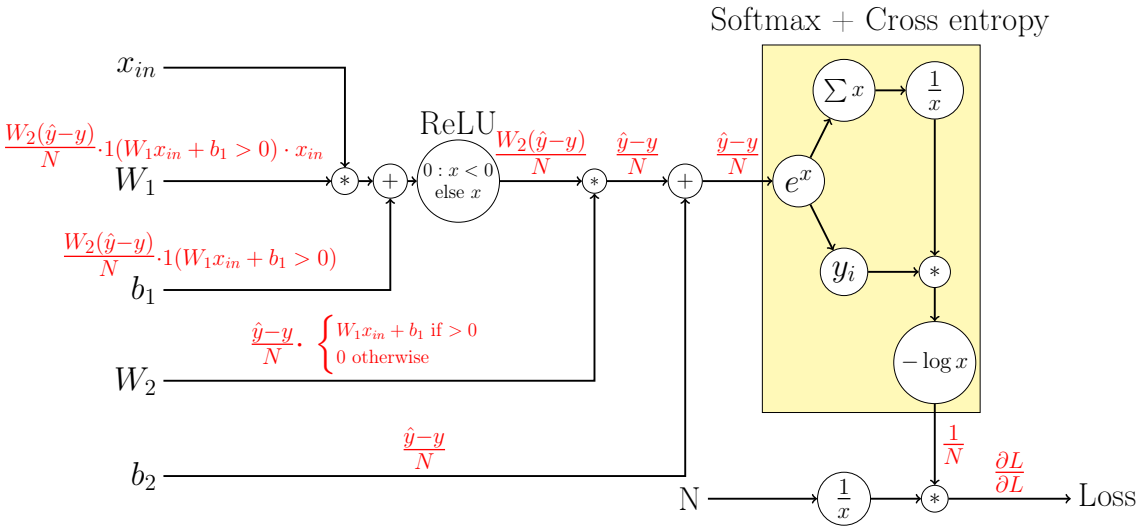


Figure 8: Backpropagation for a 2 layer neural network. The red text is the backpropagated value for each block.

In appendix G the derivative of softmax and cross entropy loss, when combined into one block, is found:

$$\frac{\partial L}{\partial \boldsymbol{s}} = \hat{\boldsymbol{y}} - \boldsymbol{y} \tag{10}$$

From the example of performed backpropagation for a simple two layer neural network

with parameters: $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$, the gradients can be written as:

$$\frac{\partial L}{\partial \boldsymbol{W_1}} = \frac{\boldsymbol{W}_2(\hat{\boldsymbol{y}} - \boldsymbol{y})}{N} \cdot 1\left(\boldsymbol{W}_1 \boldsymbol{x}_{in} + \boldsymbol{b}_1 > 0\right) \boldsymbol{x}_{in} \tag{11}$$

$$\frac{\partial L}{\partial \boldsymbol{W_2}} = \frac{\hat{\boldsymbol{y}} - \boldsymbol{y}}{N} \cdot \begin{cases} \boldsymbol{W}_1 \boldsymbol{x}_{in} + \boldsymbol{b}_1 & \text{if} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\frac{\partial L}{\partial \boldsymbol{b_1}} = \frac{\boldsymbol{W}_2(\hat{\boldsymbol{y}} - \boldsymbol{y})}{N} \cdot 1\left(\boldsymbol{W}_1 \boldsymbol{x}_{in} + \boldsymbol{b}_1 > 0\right) \tag{13}$$

$$\frac{\partial L}{\partial \boldsymbol{b_2}} = \frac{\hat{\boldsymbol{y}} - \boldsymbol{y}}{N} \tag{14}$$

Once the gradients is calculated it becomes a problem of minimization.

4.5 Optimizers

After the calculation of the gradients it is required to target the actual objective of the training phase: minimizing the loss.

The equations for the partial derivatives of the loss function, found in section 4.4, cannot be minimized directly through the usage of gradient descent techniques. The reason is that it is very expensive to calculate the loss and gradient over the whole training data. Therefore, the training is performed in batches which will be elaborated in section 4.9. This implies that the loss and gradients is only calculated for a subset of the training data, then an update is done and a new batch is taken from the training data. The functions used to perform an update step is called an optimizer.

Since the training is done in batches the first and simplest optimizer is Stochastic Gradient Descent, SGD. The pseudocode for SGD is shown in algorithm 1.

Algorithm 1: Stochastic Gradient Descent

Requires initializaion of: α (learning rate), $\boldsymbol{W}_0$ (network parameters)

while $\boldsymbol{W}$ *not converged* **do**

1 $\quad \boldsymbol{G}_t = \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t)$ ▷ Evaluate gradients at timestep t

2 $\quad \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \boldsymbol{G}_t$ ▷ Update network parameters

end

A problem of stochastic gradient descent is saddle points. At a given saddle point or near it, stochastic gradient descent will get stuck or will be very slow, since the gradient at this point is zero.

A way of improving the performance around the saddle points is to include momentum.

The momentum is implemented as a running average of the computed gradients, this will make the momentum act as a form of velocity. The pseudocode for stochastic gradient descent with momentum is shown in algorithm 2.

Algorithm 2: Stochastic Gradient Descent With Momentum

Requires initialization of: ρ (gradient decay), α (learning rate), $\boldsymbol{W}_0$ (network parameters).

while $\boldsymbol{W}$ *not converged* **do**

1 $\quad \boldsymbol{G}_t = \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t)$ ▷ Evaluate gradients at timestep t

2 $\quad \boldsymbol{V}_{t+1} = \rho \boldsymbol{V}_t + \boldsymbol{G}_t$ ▷ Evaluate gradient velocity at timestep t

3 $\quad \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \boldsymbol{V}_{t+1}$ ▷ Update network parameters

end

The addition of the momemtum will handle or at least improve the scenarios where the optimization is stuck at saddle points.

Another optimizer is AdaGrad, which uses a moment based on a running sum of the squared gradient. The pseudocode for AdaGrad is shown in algorithm 3.

Algorithm 3: AdaGrad

Requires initialization of: α (learning rate), $\boldsymbol{W}_0$ (network parameters).

while $\boldsymbol{W}$ *not converged* **do**

1 $\quad \boldsymbol{G}_t = \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t)$ ▷ Evaluate gradients at timestep t

2 $\quad \boldsymbol{A}_t = \boldsymbol{A}_{t-1} + \boldsymbol{G}_t^2$ ▷ Evaluate sum of squared gradients at timestep t

3 $\quad \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \frac{\alpha}{\sqrt{\boldsymbol{A}_t + \epsilon}} \boldsymbol{G}_t$ ▷ Update network parameters

end

The idea, behind the momentum and its usage in AdaGrad, is to scale the update step based on the previously calculated gradients. AdaGrad maintains a running sum of squared gradients, G_t, which it uses to scale the learning rate, α. In practice G_t will keep accumulating and the learning rate thus approaches zero, which will make the parameters, W, converge. The meaning behind scaling the learning rate with G_t is to create an adaptive learning rate. The adaptive learning rate will make parameters with a high gradient have a faster declining learning rate, while parameters with generally low gradients will have a slower declining learning rate.

The adaptive learning rate will make the algorithm perform shorter steps along fast changing dimensions and longer steps along slowly changing dimensions.

The fourth and last considered optimizer is Adam, which combines the ideas from AdaGrad

and the momentum used in SGD with momentum. This makes Adam compute individual adaptive learning rates for the parameters based on the combination of two momentum's, as stated in paper Kingma and Ba (2014).

Algorithm 4: Adam

Requires initialization of: α (learning rate), β_1 (velocity momentum decay), β_2 (squared gradients momentum decay), $\boldsymbol{W}_0$ (network parameters)

while $\boldsymbol{W}$ *not converged* **do**

1 $\boldsymbol{G}_t = \nabla_{\boldsymbol{W}} L(\boldsymbol{W}_t)$ ▷ Evaluate gradients at timestep t

2 $\boldsymbol{M}_t = \beta_1 \boldsymbol{M}_{t-1} + (1-\beta_1)\boldsymbol{G}_t$ ▷ Update velocity momentum

3 $\boldsymbol{V}_t = \beta_2 \boldsymbol{V}_{t-1} + (1-\beta_2)\boldsymbol{G}_t^2$ ▷ Update squared gradients momentum

4 $\hat{\boldsymbol{M}}_t = \frac{\boldsymbol{M}_t}{1-\beta_1^{t+1}}$ ▷ Bias correction

5 $\hat{\boldsymbol{V}}_t = \frac{\boldsymbol{V}_t}{1-\beta_2^{t+1}}$ ▷ Bias correction

6 $\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \frac{\hat{\boldsymbol{M}}_t}{\sqrt{\hat{\boldsymbol{V}}_t + \epsilon}}$ ▷ Update network parameters

end

The pseudocode for Adam is illustrated in algorithm 4. On line 4 and 5 it is seen that for both moment estimates there are performed bias correction. This is done since the estimates will be strongly biased towards the initial values of the moment estimates. From the combination of both estimates Adam is well suited for handling problems as saddle points and for dimensions changes at different paces.

The paper Kingma and Ba (2014) proposes the following initialization of the hyperparameters of Adam when being used for machine learning: learning rate, $\alpha = 0.001$, exponential decay rates for estimates of moments, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and to prevent division by zero in the update rule of Adam, $\epsilon = 10^{-8}$.

Based on the results presented in the paper Kingma and Ba (2014) it was chosen to use Adam as optimizer.

4.6 Initialization of Network Weights

During the explanation of the optimizers it is seen there is a requirement of the weights being initialized. The naive initialization of the weights is to set all equal to 0, but this initialization introduces a problem. The problem introduced can be seen from equation 12, where the gradients will evaluate to 0 and therefore no update step will be performed.

The other considered method is Keras' default initializer Xavier uniform. It was chosen to use Xavier uniform based on the results in the paper Glorot and Bengio (2018), where it is

shown that the outputs of the layers is becoming increasingly more drawn towards 0, the deeper the layer is placed, when using a gaussian initialization. It is a problem that the output converges towards 0, since the same inverse tendency can be seen for the gradients, which at the earlier layers goes towards 0, and therefore no update steps is performed. This problem is improved by initialization with Xavier uniform, which keeps an acceptable variation around 0 for the output of the layers and the gradients as shown in the paper Glorot and Bengio (2018).

4.7 Layers of Neural Networks

Until now all the foundational elements of neural networks have been explained. The next subject of interest is the architecture of a neural network, which includes the relevant layers used in neural network models.

4.7.1 Fully Connected Layer

The first and most simple layer is a fully connected layer. The operation of a fully connected layer is a linear operation, where every input is connected to every output, which is illustrated in figure 9.

In practice a fully connected layer can introduce a very high memory requirement, which can be shown through a simple example. If the input is in form of an 224x224 image with 3 channels, and this is to be fully connected to an output of size 1024, the amount of weights would be: $224 \cdot 224 \cdot 3 \cdot 1024 \approx 154.14 \cdot 10^6$. This is roughly equivalent to a memory usage of 616 mega bytes if the datatype is an 8-byte float.

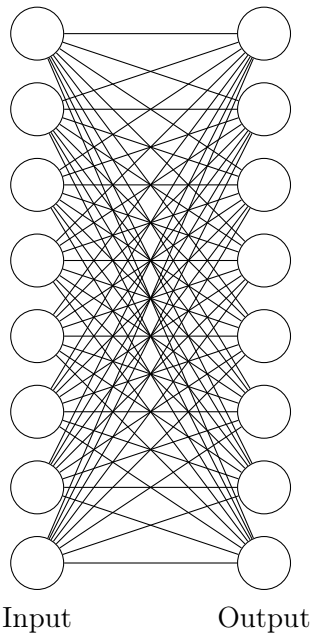


Figure 9: Fully connected layer with 10 input and output neurons.

4.7.2 Convolutional Layer

The convolutional layer is quite different from the fully connected layer. The convolutional layer is the core element of a convolutional neural network. A convolutional layer consists of N filters. These filters are often spatially small relative to the activation map that is given as input. The convolutional filter is applied at every pixel in the input map. The convolution operation is visualized in figure 10.

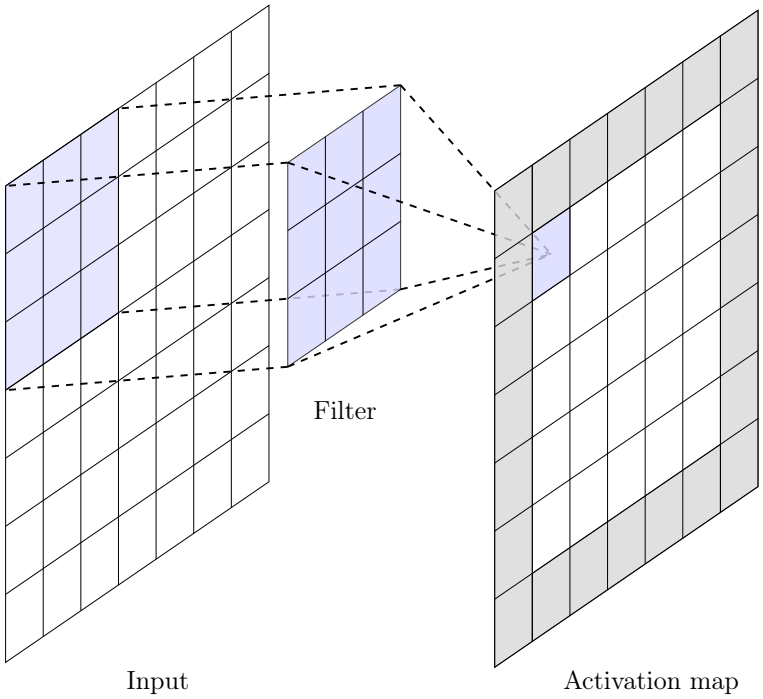


Figure 10: The convolution of a 3x3 filter in a convolutional layer

When the filter is applied it is convolved with the input, calculating the dot product at every location and generating an activation map for each filter. The depth of the filters is equal to the depth of the input. This means that if the input is an image with 3 channels, and 32 5x5x3 filters are applied, 32 activation maps will be generated. In the mentioned example, there will be the following amount of parameters, that needs to be learned:

$$5 \cdot 5 \cdot 3 \cdot 32 + 32 = 2432 \tag{15}$$

Where the last addition of 32 is the biases of each filter.

When the convolutional layer is trained each filter will learn to activate when seeing some specific visual feature. The visual features, that are recognized in each layer, rises in complexity through the network. Thereby, the layers closest to the end of the network will activate upon the most complex features. The visual features in the start of the network could be corners, edges or blobs of color, while the deeper layers could activate upon complex features like specific patterns or similar.

4.7.3 Pooling Layer

A pooling layer is used to reduce the spatially size independent of the channels. By doing this there will also follow a reduction in the amount of parameters and complexity of the

network. This might help to counteract overfitting explained in section 4.11.

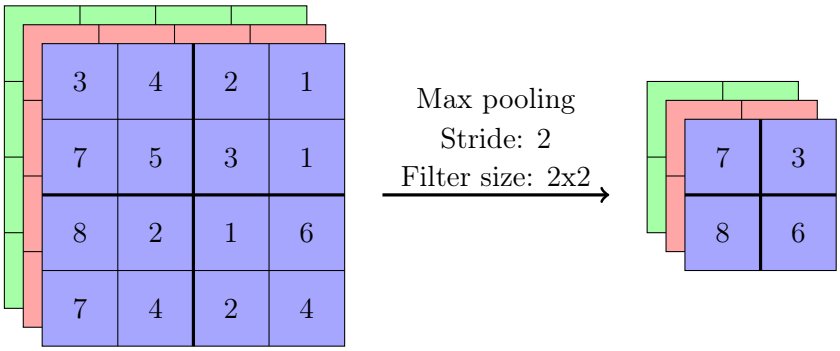


Figure 11: Max pooling operation visualized

Pooling can be performed by inserting a max pooling layer of size 2x2 with a stride of 2 as illustrated in figure 11. By inserting this max pooling layer the information flowing through the network is reduced by 75%.

4.8 Choosing a Baseline Model

Once the core elements and the architecture of the neural networks have been set up, it is needed to have some sort of reference to compare the performance of the networks. A baseline model is a very simple model, that is used for comparison of models. A model is considered acceptable if the achieved accuracy is higher than the baseline's.

The first baseline model considered is **Random Choice**, this model randomly chooses a prediction.

$$f(\boldsymbol{x}) = \frac{1}{N(\boldsymbol{x})} \sum_{i=0}^{n-1} N(\boldsymbol{x}_i) \cdot \frac{N(\boldsymbol{x}_i)}{N(\boldsymbol{x})}, \qquad i = 0, ..., n-1 \tag{16}$$

$\boldsymbol{x}$ represents the entire dataset, where $\boldsymbol{x}_i$ represents the data for the i'th class. N is the amount of a given collection. The theoretical accuracy of random choice on the large dataset is 0.1271.

The second model is **Zero Rule** which is described in the paper Devasena et al. (2011). This model always predicts the label that is most present in the training data. The accuracy of Zero Rule is given by:

$$f(\boldsymbol{x}) = \frac{N(\boldsymbol{x}_M)}{Total}, \qquad M = \arg\max_i N(\boldsymbol{x}_i) \tag{17}$$

Where M is the index of the most frequent label in the data. The accuracy for the large dataset is 0.1419. Based on the theoretical calculated accuracies it was chosen to use the

Zero Rule baseline model, since it achieves the highest theoretical accuracy.

The Zero Rule baseline is used to compare and show if the performance of a given method is considered acceptable. Another comparison that is interesting is to see if a given method is more accurate than a human. Human accuracy on the large dataset was found to be 0.9597 in appendix E.

4.9 Training a Neural Network

Before moving forward to using neural networks the training phase is needed to be clarified. When a neural network is being trained, there are several factors that requires attention, these factors will be described in this section.

4.9.1 Batch size

When training the neural networks the data is organized in batches. This implies, that the weights of a neural network is not updated after each image but after each batch. Fundamentally, there are three ways to set the batch size.

1. **Batch training**: setting the batch size equal to the size of the training data.
2. **Stochastic training**: setting the batch size equal to 1.
3. **Mini batch training**: setting the batch size to a number between 1 and the size of the training data.

When using batch training, the batch size is set equal to the size of the training data. This requires that all the data can be present in memory. When training, using this method, the loss and parameter gradients are calculated over the whole dataset, and only performs one update step after each evaluation of all data.

The second option is to set the batch size equal to 1, this would result in each image having a high impact on the updates of the weights. Furthermore, using a batch size of 1 implies that an update of the weights is needed after each image, thereby limiting the parallelism. This can make the training phase extraordinary time consuming.

The last option is using a batch size between 1 and the size of the training data, this is also called using a mini batch. The paper Masters and Luschi (2018) shows that when training the ResNet-50 on the ImageNet dataset over a continuous range of learning rates, a batch size of 32 shows the highest performance. Based upon these results it was chosen to use mini batch training, with a batch size of 32, when training the neural networks.

4.9.2 Epochs

During training, the training data is randomly put in batches. The neural network is trained on all the batches and performs an update step after each batch. A full iteration over all the batches is called an epoch. If the amount of epochs is set too low, the network might not reach its full potential. When choosing the amount of epochs, that should be used, the performance metrics of the neural network should be inspected as an indication of when the network's performance starts to stagnate.

4.9.3 Performance Metrics

When evaluating a neural network for classification, the metrics used is loss and accuracy. The loss was described in section 4.2 as used for optimization of the network's weights. Besides this usage, the loss is also used to interpret the networks performance.

When analyzing the performance of a neural network, the metrics is calculated both for the training data and the validation data. By calculating the metrics for the validation data, it is possible to see how, the network performs on unseen data compared to the data it is training on. Furthermore, by evaluating the validation accuracy, it is possible to save the parameters after each epoch, if it achieves a higher validation accuracy than earlier evaluations during the training. This results in the best parameters with respect to the validation accuracy, found through the training phase, to be saved.

4.10 Data Preprocessing

Before the data is presented to the neural networks, multiple preprocessing techniques can be applied, to make the data representation easier to interpret for the neural networks. The considered preprocessing methods is image rescaling and mean subtraction.

4.10.1 Image rescaling

The first data preprocessing method is image rescaling, which rescales all the image's pixel values. In the paper Kotsiantis et al. (2006), it is stated that normalization of the input data is important for neural networks. One of the reasons behind this is, that when the input is large, the error will be correspondingly large and then might make the learning phase unstable, when performing large steps during the updates. It was chosen to perform rescaling, making the input values vary in the interval $[0,\ 1]$.

4.10.2 Mean subtraction

The last considered preprocessing method is to subtract the mean value from all the images to centralize the data around origo, which might make the network less sensitive to changes in weights. The centering around origo through mean subtraction is illustrated in figure 12, where the line can be seen as a simple linear classifier splitting two categories: blue circles and red rectangles.

(a) Data points before mean subtraction. (b) Data points after mean subtraction.

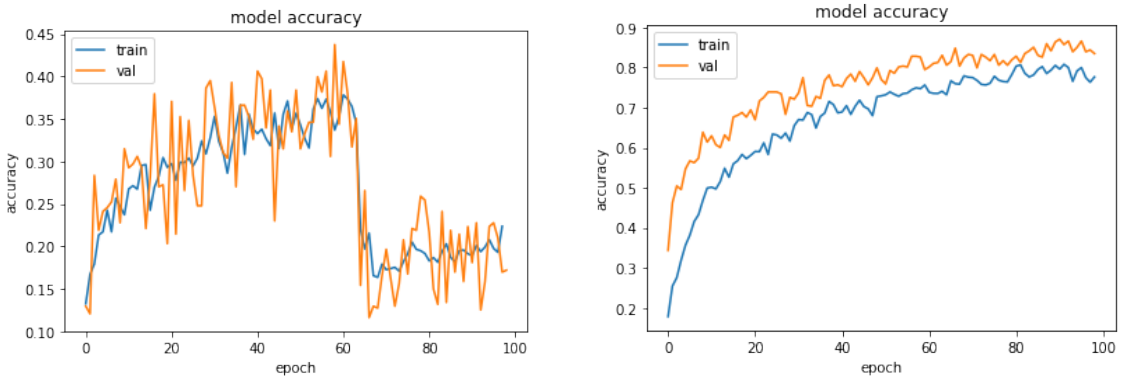
Figure 12: Simple illustration of data points partition before and after mean subtraction

In figure 12a, it is shown that without mean subtraction small variations in the linear classifier will have a large impact on the predictions. The opposite is shown in figure 12b, where the data is centered around origo.

Regarding the mean subtraction, two different ways of performing this is considered: mean image subtraction and per channel mean subtraction.

Mean image subtraction calculates a mean image based on all the images in the training data. This implies each pixel in the training data is averaged individually.

The second method considered is per channel mean subtraction, this is the default mean subtraction used by the neural network library Keras. When performing per channel mean subtraction, only one mean value is calculated for each channel. Thus, there is in contrary to the mean image subtraction less information in calculation of the means. Per channel mean subtraction is less computationally heavy compared to the mean image subtraction, but introduces instability in the training phase.



(a) Plot of accuracy for a 2 layer fully connected neural network, when using per channel mean subtraction during training

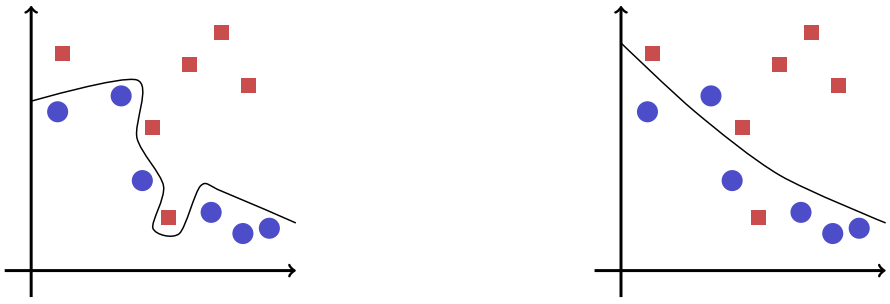
(b) Plot of accuracy for a 2 layer fully connected neural network, when using image mean subtraction during training

Figure 13: Visualizing the instability, introduced when using per channel mean subtraction, compared to image mean subtraction a 2 layer fully connected neural network.

Based on the instability introduced by per channel mean subtraction, shown in figure 13a, it was chosen to choose mean image subtraction, when performing mean subtraction.

4.11 The Problem of Overfitting

An essential problem regarding neural networks is overfitting. The phenomenon of overfitting a neural network, is when the network becomes too accurate with respect to the training data. This will most likely make the neural network perform worse on unseen data. It is relevant to look at overfitting in this bachelor, since the network is trained on a relatively small dataset consisting of a total of 2347 images, where the training data only consists of 1875 images.



(a) Overfitted classifier

(b) Desired classifier

Figure 14: Simplified illustration of overfitting

A simplified illustration of overfitting is shown in figure 14, where it is seen in figure 14a, that when a classifier is overfitted it learns the training data very well, but have a hard time

recognizing the trend in the data. While it is seen in figure 14b, that a desired classifier learns the trend in the data and uses this trend to classify the data.

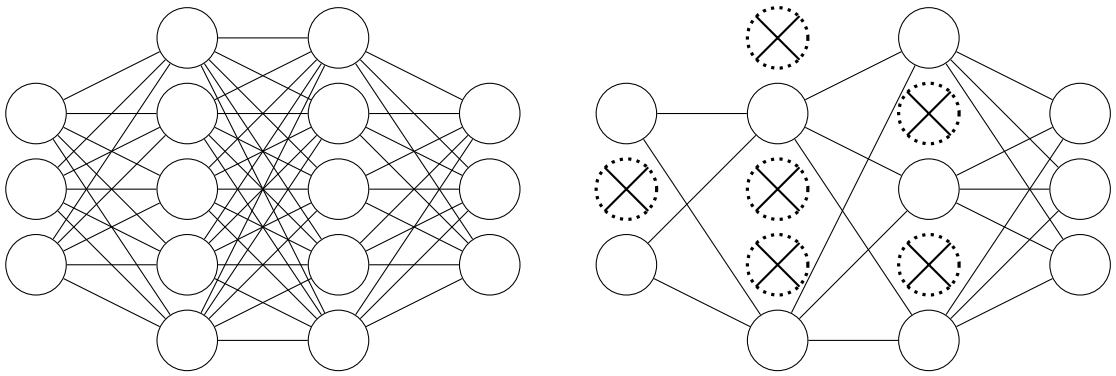
To counteract overfitting regularization is introduced, which will be explained before moving into two essential regularization techniques.

4.11.1 Regularization

When counteracting overfitting of neural networks regularization can be introduced in the network. Regularization can be implemented in various forms, this includes adding additional layers like dropout, changing images as part of preprocessing, or reducing the information flowing through the network.

Dropout

Dropout is an addition to the network, which is only present during the training phase. Dropout can be seen as a temporary layer, that ignores neurons by temporary cancelling their incoming and outgoing connections in the neural network.



(a) Neural network (b) Neural network with dropout

Figure 15: Illustrating the effect of dropout

Figure 15 illustrates the deactivation of neurons in a fully connected neural network. When dropout is applied in figure 15b, the resulting connections can be seen as a sub-net of the original neural network seen in figure 15a. The deactivation of a neuron during training is based on an independent probability, which is suggested to be set equal to 0.5 in the paper Srivastava et al. (2014).

Data augmentation

Data augmentation is a preprocessing technique used to make the dataset appear larger. This is done by performing various transformations on the training data. Since the dataset

is rather small, this technique might show useful.

The augmentation factors, which are considered:

1. Change in brightness
2. Vertical flip
3. Horizontal flip
4. Rotation

By combining the various factors, the input images can be altered slightly, but to the network seem like completely different images, thereby artificially enlarging the dataset.

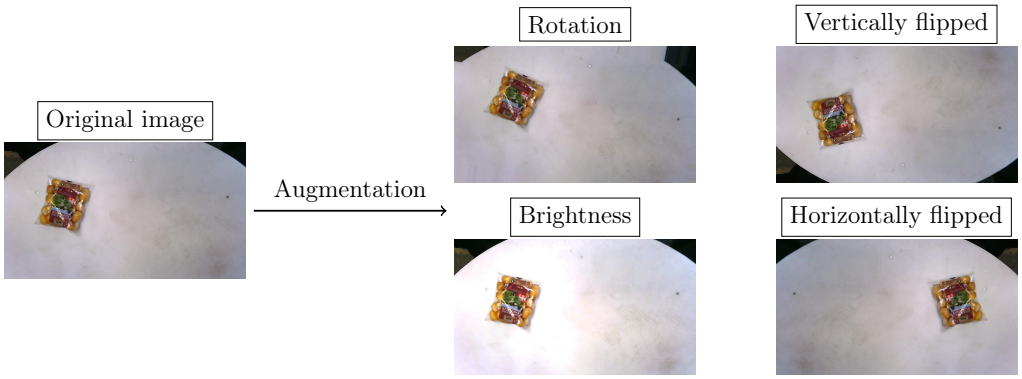


Figure 16: Data augmentation and the used augmentation factors visualized

An example of the augmentation factors is shown in figure 16. When the augmentation factors are applied, it is randomly chosen, which factors should be applied and to which extend they should be applied. The range of values for which the images can be rotated is within the interval $[-10,\ 10]$ degrees. This range of values were chosen, since none of the objects were rotated out of the image. The brightness range is in the interval $[0.65,\ 1.35]$, this means that the brightness can be made 35% relatively brighter or darker. These values were chosen as they represented realistic variations in brightness.

5 Methods for Classification of Consumables

The problem of classification is to connect an image to a well defined label. This can then be used in the robot cell to determine how to act, based upon the label applied to the image.

In this section the classification will be applied on a full sized image, where the objective is to classify whether or not there is any of the classes present in the image. Examples of the images, that are used in this section, is shown in section 3, figure 3. The image is classified into one of the following 8 classes: *potato*, *table*, *cat food salmon*, *cat food beef*, *ketchup*, *arm*, *carrots* and *buns*.

Throughout this section different methods will be designed, and at the end they will be evaluated and compared.

5.1 k-Nearest Neighbor

The first method considered is a simple method called k-Nearest Neighbor, kNN. The principle behind this method is that all the training data is memorized and not learned.

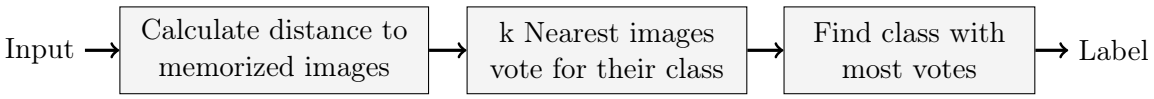


Figure 17: The prediction workflow of kNN illustrated

The prediction stage of kNN is illustrated in figure 17. When kNN is predicting it needs some kind of distance metric, which can quantify how far the input image is from the memorized images. After the distance evaluations, the closest k memorized images vote for their class, the label corresponding to the class with most votes is assigned as label to the input image.

The distance metric used is the euclidean distance, defined as:

$$d(\boldsymbol{I_1}, \boldsymbol{I_2}) = \sqrt{\sum_{i,j=0}^{rows,cols} \left(\boldsymbol{I_1^{(i,j)}} - \boldsymbol{I_2^{(i,j)}}\right)^2} \tag{18}$$

The matrices $\boldsymbol{I}$ contains some feature information from the two images, that are being evaluated. A common feature to compare images with, when using kNN is their pixel RGB intensity.

A drawback of using the pixel RGB intensity is, that the distance metric is made very

vulnerable to the localization of objects in the images. This is because the pixels are compared directly, therefore the spatial meaning of the features is kept.

A way to draw out relevant features from an image, and removing the spatial meaning of these is by calculating a histogram of the images' channels. This will make the distance metric invulnerable to localization, but solely rely upon the color of the objects.

To show the effects of using pixel values as input feature versus histogram as input feature to the matching metric, 3 images containing the same class are evaluated, these are shown in figure 18. The image in figure 18a is used as a reference image, while the image in figure 18b needs to be very similar to the reference image. The last image in figure 18c is needed to be quite different.

(a) Reference image of an potato

(b) Image of a potato very similar to the reference image

(c) Image of a potato more different from the reference image

Figure 18: Images used for showing the differences between using pixel values or histogram as input to distance metric

The results from applying the matching metric on the images is shown in table 1.

Input feature	**Distance between (a) and (b)**	**Distance between (a) and (c)**	**Relative difference between calculated differences**
Pixel value	10380.87	17037.95	0.39
Histogram	0.3444	0.4592	0.25

Table 1: Calculated differences using the difference metric for the images in figure 18

In table 1, the relative difference between the two calculated distances is significantly lower, when using a histogram as input feature. This is favorable, since the distance should be as low as possible, when the distance is calculated between two images containing the same class.

When using kNN multiple parameters should be considered and chosen prior to the usage of the method, these parameters are called hyperparameters. This includes the value of k, how many of the closest neighbors that are considered when voting. Furthermore, the preprocessing of the images before given to kNN should also be considered, since these also have an impact on the performance. The following preprocessing methods is considered:

resizing of the images, gaussian smoothing and multispectral band thresholding.

5.1.1 Hyperparameter Optimization for kNN

When the hyperparameters are optimized for kNN it is done in sequence, meaning that only one hyperparameter is optimized at a time. The first hyperparameter tuned is the value of k. This is done using cross validation, illustrated in figure 19, which is an optimization technique, that is relevant, when the training data is limited. When five fold cross validation is performed, the hyperparameter is trained 5 times, switching which fold acts as validation data. The results is then meaned over the 5 trainings, and tested on the test data to indicate the performance of kNN on unseen data. The values of k used in the optimization is: $[1, 3, 5, 8, 10, 12]$.

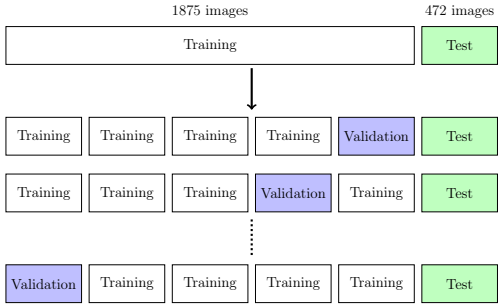


Figure 19: Five fold cross validation.

The other hyperparameters is tuned without the usage of cross validation. The hyperparameters and their values are stated in table 2.

Input feature	**Resizing**	**Bin size**	**Gaussian smoothing**	**Multispectral band thresholding**
Pixel value	32x32	8x8x8	False	False
Histogram	64x64	16x16x16	True	True
	128x128	32x32x32		

Table 2: Hyperparameters and their values tested for kNN

The hyperparameter "Resizing" is only used, when the input feature is pixel values, while the dimensions of the input is kept for calculation of the histogram. The hyperparameter "Bin size" is only used, when the input feature is histogram, this is used to quantify the number of bins of the 3 channels in the histogram.

The problem of applying the multispectral band tresholding, is that the kNN method is made vulnerable to its environment. If the memorized images is taken from the large dataset, but the kNN method then is used on images taken in another environment, the multispectral band tresholding might not filter away the environment and thus might not work as intended.

Since the hyperparameters have been chosen, hyperparameter optimization can now be performed. Because of the problem involved with thresholding the images, it was chosen to present the best results with and without usage of thresholding in table 3.

Summary	Prediction time [ms]	Memory usage [MB]	Test accuracy
With thresholding	24.4	7.68	0.9560
Without thresholding	569	491.52	0.9000

Table 3: The best results for kNN with and without multispectral band thresholding

The results for the other combinations of hyperparameters is shown in appendix F. The hyperparameters for the two best kNNs is shown in table 4.

Summary	Input feature	Bin size	Gaussian smoothing	k
With thresholding	Histogram	8x8x8	False	1
Without thresholding	Histogram	32x32x32	False	1

Table 4: The hyperparameters for the best results of kNN with and without the usage of multispectral band thresholding.

It is seen that for both with and without multispectral band thresholding, the input feature is histogram and there is no application of gaussian smoothing. Furthermore, it can be seen that the value of k is 1, which means that kNN might be very vulnerable to outliers in the data.

5.2 Designing Neural Networks

In section 4, the background knowledge of neural networks was explained. In this section, the theory will be used to design networks targeted at solving the classification problem.

The first neural network, that will be designed is a Fully Connected Network, FCN, which consists solely of fully connected layers. The second type of neural network designed is a Convolutional Neural Network, CNN, which consists of multiple types of layers. It was chosen to resize the input images to the neural networks into a size of 224x224, which is an often used image size for neural networks.

5.2.1 Fully Connected Neural Network

A fully connected network requires the input to the fully connected layers to be a vector. Therefore, the first that happens to an image, when fed into a FCN, is flattening. A summary of the designed FCN is shown in table 5.

	Layer	Size	Activation
Input	Image	224x224x3	-
1	Flatten	-	-
2	Fully connected	1xF	ReLu
3	Fully connected	1x8	Softmax

Table 5: Summary of the designed fully connected network.

In table 5 there is a missing quantity, F, which is the size of the first fully connected layer. The size of this layer is to be determined during hyperparameter optimization in section 5.4.

5.2.2 Convolutional Neural Network

The designed convolutional neural network, is defined from a block structure, illustrated in figure 20.

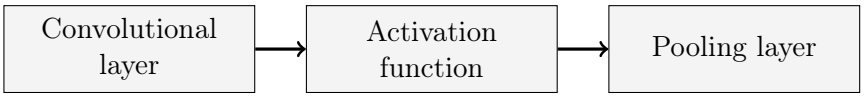


Figure 20: Illustration of the block structure used in the design of CNNs.

The CNN is constructed from an given amount of the above structural block in sequence, while the end of the network is connected to fully connected layers. The fully connected layers in the end acts as a classifier, based on the features which have been extracted by the convolutional blocks. A summary of the designed CNN with 2 blocks is shown in table 6.

	Layer	Size	Filter amount	Activation
Input	Image	224x224x3	-	-
1	Convolution	NxNx3	32	ReLu
2	Max Pooling	MxM	-	-
3	Convolution	NxNx32	64	ReLu
4	Max Pooling	MxM	-	-
5	Flatten	-	-	-
6	Fully Connected	1x32	-	ReLu
7	Dropout	-	-	-
8	Fully Connected	1x8	-	Softmax

Table 6: Summary of a designed 2-block CNN

It is seen in the summary of the designed 2-block CNN, that there is multiple variables in the size of the layers, these will be determined during the hyperparameter optimization in section 5.4.

The choice of the amount of filters is inspired from the VGG16 network Simonyan and Zisserman (2014), which has a telescoping structure in the sense, that the amount of filters doubles throughout the network.

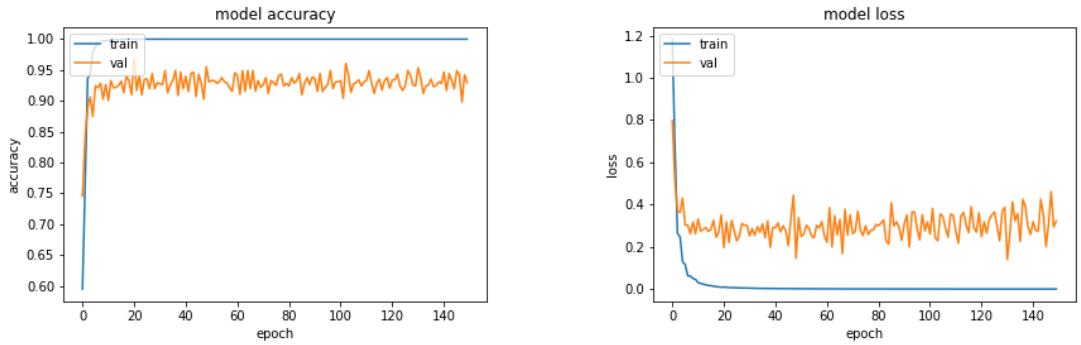
Besides the 2-block CNN, shown in table 6, a 3-block CNN is also designed, with the same type of hyperparameters, namely the filter size of the convolutional layers, the size of the second last fully connected layer and the size of the max pooling layers. The amount of filters in the last convolutional block of the 3-block CNN was chosen to be 128, keeping the telescoping structure of the filter sizes.

5.3 Investigating Overfitting in the Designed Networks

Before appliying the regularization techniques, described in section 4.11, the accuracy and loss graphs from a training session without regularization is inspected. From these graphs it can be seen if a network overfits, and thus if it needs addition of regularization.

Inspection of the Designed Fully Connected Network

The FCN is trained on the large dataset yielding the graphs in figure 21.

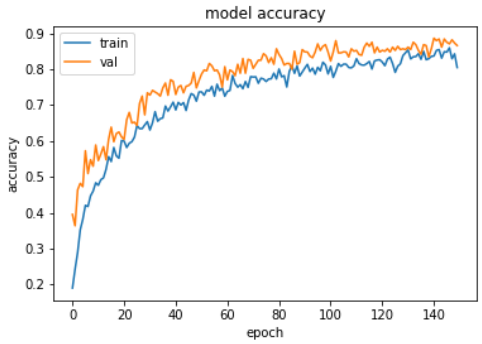


(a) Accuracy versus epochs for the FCN without regularization on the large dataset

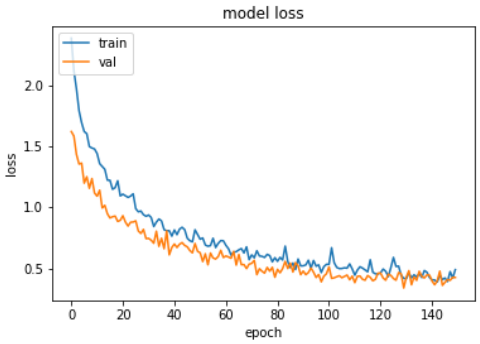
(b) Loss versus epochs for the FCN without regularization on the large dataset

Figure 21: Training metrics for the FCN without regularization on the large dataset

Figure 21 shows the FCN is overfitting, because the accuracy of the training data is significantly higher than the validation accuracy. Furthermore, the loss of the training data is significantly lower than the validation loss. It is seen that the network obtains a high validation accuracy, which could be because the network indirectly overfits to the validation data. If the network overfits the data, it is very vulnerable to variations in the data. Therefore, it was chosen to add regularization in form of data augmentation to the network.



(a) Accuracy versus epochs for the FCN with data augmentation on the large dataset



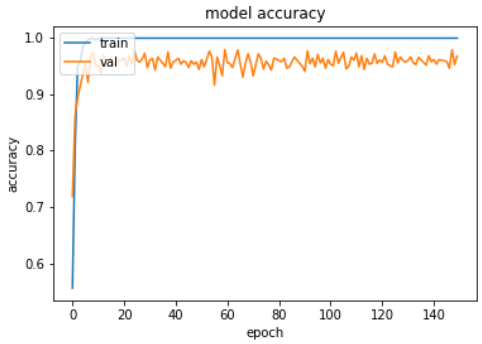
(b) Loss versus epochs for the FCN with data augmentation on the large dataset

Figure 22: Training metrics for the FCN with data augmentation on the large dataset

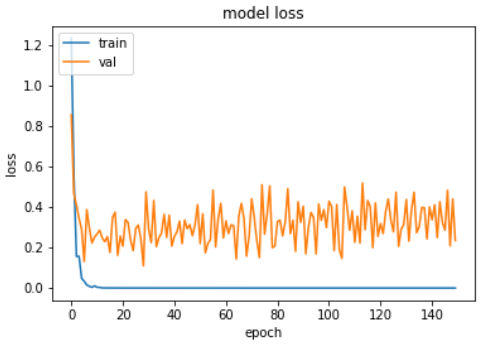
From the graphs in figure 22, it is noticed that the addition of data augmentation in the data preprocessing counteracts the overfitting. Based on this, the addition of data augmentation is deemed as enough regularization for the FCN.

Inspection of the Designed 2-Block CNN

The 2-Block CNN already contains some regularization, which comes from the design of the model in form of pooling layers. The pooling layers limits the information flow through the network and thus lowers the capacity of the network. The 2-Block CNN is now trained without additional regularization on the large dataset.



(a) Accuracy versus epochs without regularization for the 2-Block CNN on the large dataset

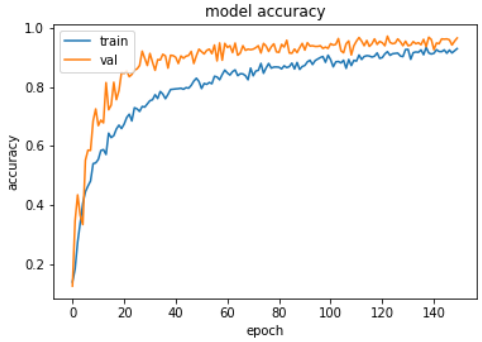


(b) Loss versus epochs without regularization for the 2-Block CNN on the large dataset

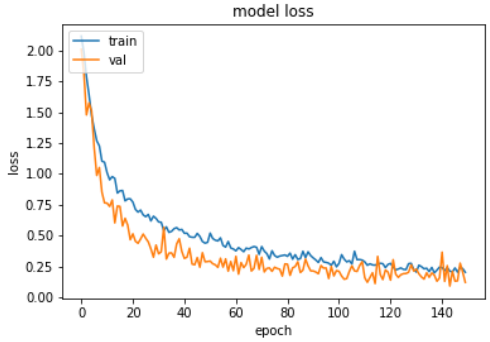
Figure 23: Plot of accuracy and loss without regularization for the 2-Block CNN on the large dataset

Figure 23 shows clear signs of overfitting. It is seen that the network has a hard time generalizing what is learned from the training data to the validation data.

Dropout is now added between the last two fully connected layers, as seen in table 6. Furthermore, data augmentation is now performed during preprocessing.



(a) Accuracy versus epochs with dropout and data augmentation for the 2-Block CNN on the large dataset



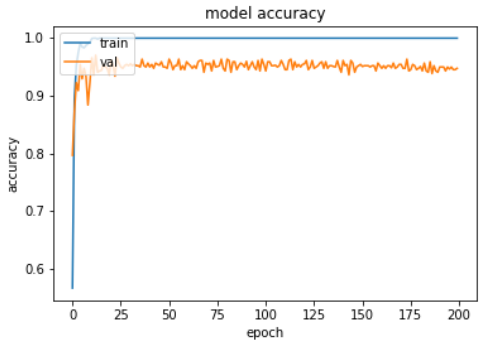
(b) Loss versus epochs with dropout and data augmentation for the 2-Block CNN on the large dataset

Figure 24: Plot of accuracy and loss with dropout and data augmentation for the 2-Block CNN on the large dataset

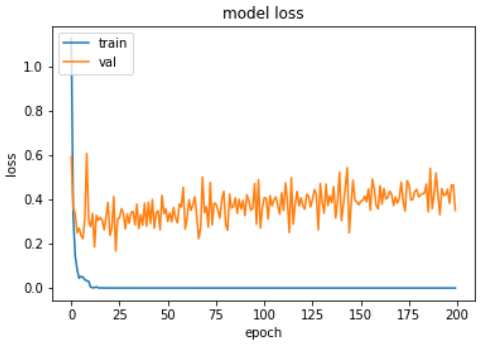
In figure 24 all signs of overfitting were removed by the introduction of data augmentation and dropout. This indicates that the introduced regularization, in form of data augmentation and dropout, is fine for the 2-block CNN. In addition to the removal of overfitting, the validation accuracy rises significantly above the training accuracy. This might be because of the introduction of dropout in the model, which makes the neural network during training act like a model ensemble, consisting of multiple weaker networks, while during validation they are combined into one strong network.

Inspection of the Designed 3-Block CNN

The 3-Block CNN is trained on the large dataset without any additionally regularization than the pooling layers. When trained on the large dataset the following training metrics is obtained.



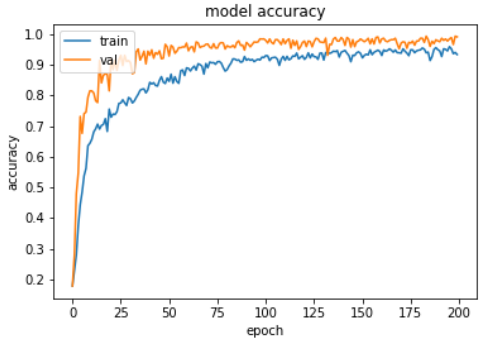
(a) Accuracy versus epochs without regularization for the 3-Block CNN on the large dataset



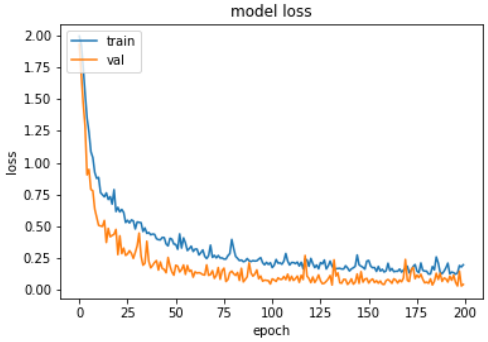
(b) Loss versus epochs without regularization for the 3-Block CNN on the large dataset

Figure 25: Plot of accuracy and loss without regularization for the 3-Block CNN on the large dataset

In figure 25 overfitting is present, therefore data augmentation and dropout is applied in the same manner as for the 2-Block CNN.



(a) Accuracy versus epochs with dropout and data augmentation for the 3-Block CNN on the large dataset



(b) Loss versus epochs with dropout and data augmentation for the 3-Block CNN on the large dataset

Figure 26: Plot of accuracy and loss with dropout and data augmentation for the 3-Block CNN on the large dataset

When training the 3-Block CNN, with the addition of dropout and data augmentation, it is seen from figure 26 that overfitting is counteracted.

5.4 Hyperparameter Optimization

The final stage, of the neural network design procedure, is to perform hyperparameter optimization. Hyperparameters are all the parameters which are set prior to the learning phase, and thus not learnable. This includes the learning rate of the optimizer, the amount of epochs for training, the size of the layers and several other.

When hyperparameter optimization is performed, it can be seen as a search in the parameter space for the optimal combination of hyperparameters. It would be impossible to evaluate all combinations of hyperparameters, therefore values are selected based on trial and error.

The hyperparameters that are being tuned in the FCN is the learning rate for the optimizer and the size of the fully connected layer, which is shown in table 7 below.

Learning rate	**Size of fully connected layer**
1e-3	32
1e-4	64
	128
	256
	512

Table 7: Hyperparameters for the FCN and the values at which they are tested.

The hyperparameters for the 2-Block CNN, that is being tuned, is the learning rate, the size of the convolutional filters and the size of the max pooling layer shown in table 8.

Learning rate	**Size of convolutional filters**	**Size of max pooling**
1e-2	3x3	2x2
1e-3	5x5	4x4
1e-4		

Table 8: Hyperparameters for the 2-Block CNN and the values at which they are tested.

When performing hyperparameter optimization, the networks are trained on the training data and evaluated on the validation data. If a combination of parameters is found to have a higher validation accuracy, than the previously highest combination of parameters, then it is saved.

From the results of hyperparameter optimization shown in appendix D, the final models can now be stated. The learning rate of the FCN is found to be $\alpha = 1e - 4$ and the size of the first fully connected layer is found to be $F = 64$, which yields the structure shown in table 9.

	Layer	Size	Activation
Input	Image	224x224x3	-
1	Flatten	-	-
2	Fully connected	1x64	ReLu
3	Fully connected	1x8	Softmax

Table 9: Summary of fully connected network with hyperparameter values filled in.

The learning rate of the 2-Block CNN is found to be $\alpha = 1e-3$, the convolutional filter size is found to be $N = 3$ and the size of the max pooling operation is $M = 2$. This yields the structure shown in the table 10.

	Layer	Size	Filter amount	Activation
Input	Image	224x224x3	-	-
1	Convolution	3x3x3	32	ReLu
2	Max Pooling	2x2	-	-
3	Convolution	3x3x32	64	ReLu
4	Max Pooling	2x2	-	-
5	Flatten	-	-	-
6	Fully Connected	1x32	-	ReLu
7	Dropout	-	-	-
8	Fully Connected	1x8	-	Softmax

Table 10: Summary of 2-block CNN with hyperparameter values filled in.

The hyperparameters $M = 2$, $N = 3$ and the learning rate $\alpha = 1e-3$ for the 2-Block CNN is reused for the 3-Block CNN, which results in the structure shown in table 11.

	Layer	Size	Filter amount	Activation
Input	Image	224x224x3	-	-
1	Convolution	3x3x3	32	ReLu
2	Max Pooling	2x2	-	-
3	Convolution	3x3x32	64	ReLu
4	Max Pooling	2x2	-	-
3	Convolution	3x3x64	128	ReLu
4	Max Pooling	2x2	-	-
5	Flatten	-	-	-
6	Fully Connected	1x32	-	ReLu
7	Dropout	-	-	-
8	Fully Connected	1x8	-	Softmax

Table 11: Summary of 3-block CNN with hyperparameter values filled in.

5.5 Evaluating the Classification Methods

From the results presented in appendix D and by training the 3-Block CNN, with the hyperparameters found from hyperparameter optimization of the 2-Block CNN, a table with summaries is created:

Summary	Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
kNN	-	-	-	-	0.9000
kNN with thresholding	-	-	-	-	0.9560
FCN	142	0.4682	0.8153	0.4063	0.8772
2-Block CNN	120	0.2259	0.9096	0.1191	0.9732
3-Block CNN	184	0.1814	0.9386	0.0354	0.9932
Zero-Rule	-	-	-	-	0.1419
Human accuracy	-	-	-	-	0.9597

Table 12: Best classification accuracy results of the classification methods.

The table is constructed from the epoch during the training, where the highest validation accuracy was achieved.

From the results presented in table 12, it is noticed that all the classification methods achieves higher accuracy than the base line model, Zero Rule. Furthermore, it is seen that only the two CNNs achieves higher accuracy than the human accuracy. The addition of a third block in the CNN lowers the validation error rate by 75%.

Other relevant information to consider, before choosing a classification method, is the time it takes for a single prediction and the memory requirement of the method.

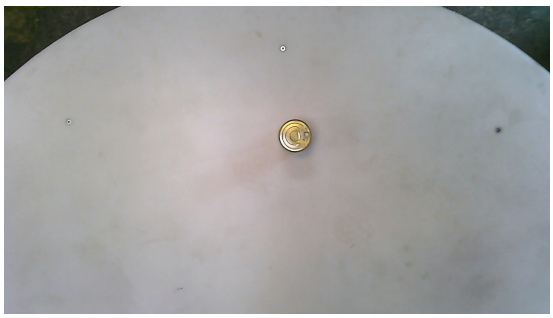
Summary	Prediction time [ms]	Memory usage [MB]
kNN	569.0	491.52
kNN with thresholding	24.4	7.68
FCN	2.0	110
2-Block CNN	1.0	69
3-Block CNN	3.0	33

Table 13: Time used for a single prediction meaned over 30 predictions and memory usage of the classification methods.

In table 13, it is seen that the computational time of both versions of kNN is higher than the computational time of the neural networks. Furthermore, the 3-Block CNN achieving the highest validation accuracy, it is also the neural network having the lowest memory requirement. Based on these results, the 3-Block CNN should be used when classifying objects in the images.

5.5.1 Problem of Predicting Similar Objects

From the results presented in table 12, it is seen that all models make some kind of error on the validation data. Some of the errors is introduced by the two classes *cat food salmon* and *cat food beef*, since they are identical, when seen directly from above. This is illustrated in figure 27.



(a) Cat food beef shown from above

(b) Cat food salmon shown from above

Figure 27: Both cat food variants seen from above, showing that there is no visible difference besides luminance, when seen from certain angles.

To show the errors introduced from having to classify two identical objects, when viewed from certain angles, the class *cat food beef* is removed from the large dataset, resulting in the dataset only consisting of 2054 images distributed among 7 classes. All methods are then trained in the same manner as described so far.

Summary	Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
kNN	-	-	-	-	0.9250
kNN with thresholding	-	-	-	-	0.9850
FCN	114	0.3972	0.8645	0.3383	0.9062
2-Block CNN	88	0.2765	0.9056	0.1327	0.9818
3-Block CNN	121	0.1224	0.9561	0.0054	1.0000
Zero Rule	-	-	-	-	0.1622
Human accuracy	-	-	-	-	0.9975

Table 14: Classification results from training the networks on the large dataset with removal of a cat food beef.

Table 14 shows that all the methods achieves higher validation accuracy, when used on the modified dataset. Furthermore, the 3-Block CNN is still the method with highest validation accuracy and is now the only method to achieve higher accuracy than the human accuracy. This indicates that the classification methods is to some degree vulnerable to similar objects, that are very hard to differ from each other, when viewed from certain angles.

6 Methods for Localization of Consumables

Classification in itself only contributes with information about what is in the image, but none information about where in the image the classified object is located. By applying localization techniques after classification, the object will be localized by finding a region of interest and assuming that the object is located in the found region of interest. Another approach is to use localization techniques as proposals and then use a classification method on the proposed region of interest.

6.1 Finding Regions of Interest

Finding Regions of Interest, RoIs, can be done by various techniques. Two method considered are **backprojection** and **background subtraction**. Before dwelling into the methods for localization, it is needed to calculate a threshold for the area that is going to be found by the localization techniques.

6.1.1 Determining Minimum Area for Regions of Interest

To determine the minimum area of a RoI, the area of a can of cat food, when seen from above, is calculated. This is done by setting up a camera matrix using the pinhole camera model. The camera matrix can be used to map 3D points in the real world to 2D points in an image.

$$\begin{bmatrix} i \cdot w \\ j \cdot w \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & c_i \\ 0 & f & c_j \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{width}{2 \cdot tan(\frac{FoV}{2})} & 0 & \frac{width-1}{2} \\ 0 & \frac{width}{2 \cdot tan(\frac{FoV}{2})} & \frac{height-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{19}$$

The camera matrix is set up based on the image indexing (i, j), depth w and spatial coordinates (x, y, z). The variables c_i and c_j is representing respectively the center of the *width* and center of the *height* of the image. Lastly, the variable FoV is the Field of View.

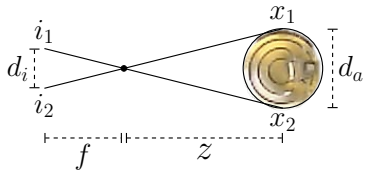


Figure 28: Pinhole model of the cat food can in the image and the real world.

The FoV is found as 75 degrees in the datasheet ELP (2013). Furthermore, a resolution of 720x1280 is used. Figure 28 illustrates the cat food can in the real world and on the image, where the center of the cat food can is at the optical center. To described the relationship between d_i and d_a, which is shown

in figure 28, the camera matrix in equation 19 is utilized.

$$\begin{aligned} i_1 \cdot w &= f \cdot x_1 + c_i \cdot z \\ i_2 \cdot w &= f \cdot x_2 + c_i \cdot z \end{aligned} \Rightarrow (i_2 - i_1) \cdot w = f \cdot (x_2 - x_1) \Rightarrow \frac{f}{i_2 - i_1} = \frac{w}{x_2 - x_1} \tag{20}$$

Figure 28 and equation 20 yields the relationship $\frac{f}{d_i} = \frac{w}{d_a}$. The distance from the camera to the table is 140 centimeter and the cat food has a height of 11 centimeter. Thus, the distance z becomes 129 centimeter. The diameter d_a is measured to 7.5 centimeter. Since d_a, f and w is known, the diameter d_i can be calculated leading to the area of the cat food can in pixels.

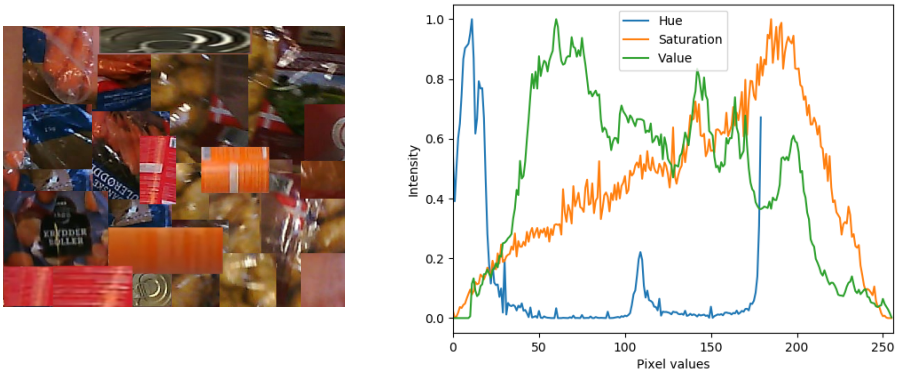
$$\frac{f \cdot d_a}{w} = d_i \Rightarrow \frac{834.0642[\text{pixels}] \cdot 7.5[cm]}{129[cm]} = 48.492[\text{pixels}] \tag{21}$$

$$\Rightarrow \pi \cdot \left(\frac{48.492}{2}\right)^2 = 1846.843[\text{pixels}^2] \tag{22}$$

Based on the calculated area of 1846.843 pixels the optimal area threshold was empirically found to be 1550 pixels.

6.1.2 Backprojection

Backprojection is a method, which is based on generation of a probability map. The probability map is generated from a histogram of a template image. The template and the histogram of the template, can be seen in figure 29.



(a) Template used for backprojection

(b) Normalized histogram of the template for backprojection

Figure 29: Template and histogram from which the probability map is created in backprojection.

The template shown in figure 29a is sampled from images containing the classes, that

should be localized. When backprojection is performed, the histogram is projected onto an input image. By doing this the normalized histogram is seen as a probability of a specific value being present in one of the objects of interest, thereby creating a probability map. The probability map can then be used to localize interesting areas, where objects of interest may be present. This can be done by finding contours in the probability map and thresholding these.

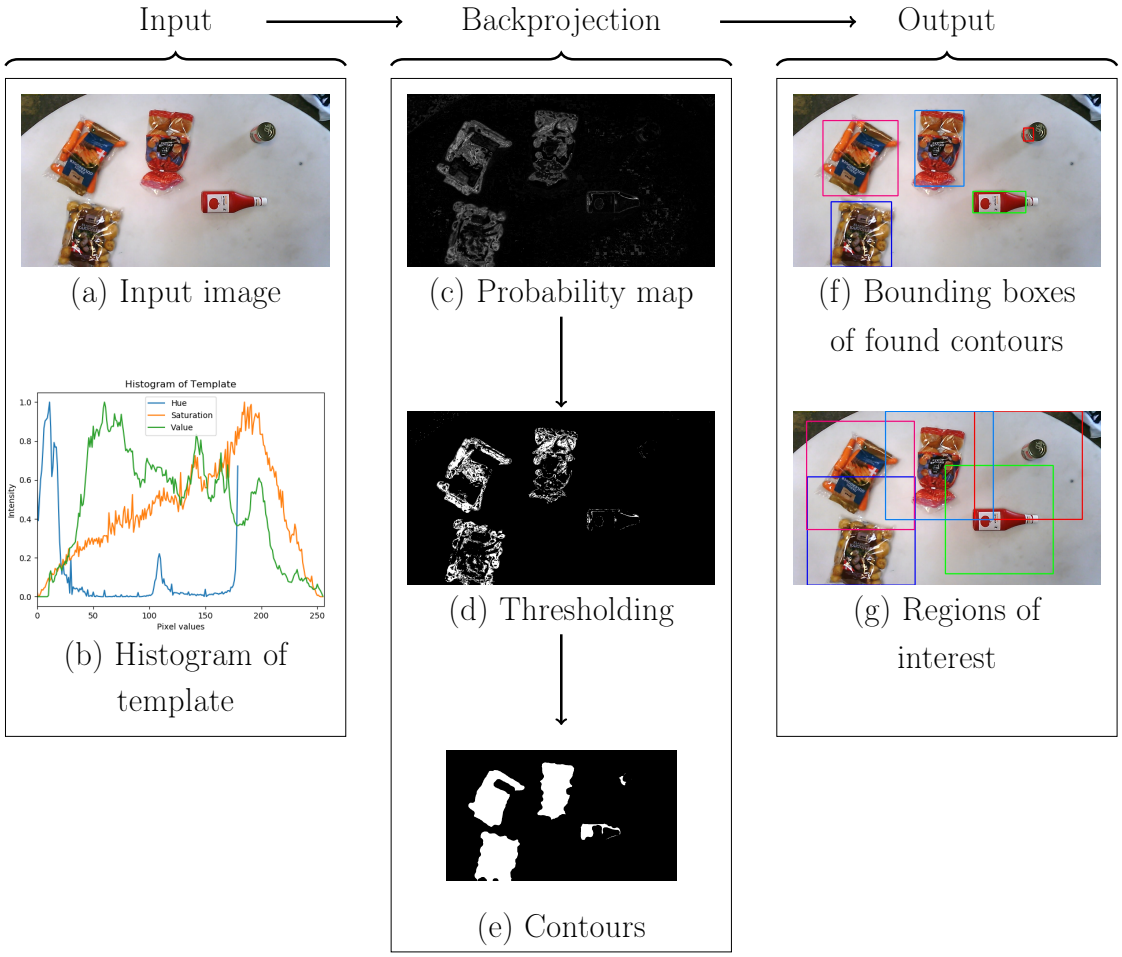


Figure 30: The Process of Backprojection

The process of the implemented backprojection is illustrated in figure 30. From the input image and histogram of the template, backprojection is performed resulting in the probability map in 30c. In figure 30d, probability thresholding is then performed, zeroing all pixels with a probability value below 0.25.

The contours is then found in figure 30e, by performing morphological closing and opening, to filter noise, followed by contour search. Finally, the found contours is thresholded based on their areas, using the calculated area threshold for RoIs from section 6.1.1.

To perform the actual localization, bounding boxes of the contours is calculated, these are shown in figure 30f. To generate a RoI, that can be used for classification of the region, an area is spatially expanded from the center of the contour up to a size of 448x448, which is illustrated by rectangles in figure 30g.

6.1.3 Background Subtraction

It was seen that backprojection is a localization technique, which is dependent on a template of sampled images of interest. Background subtraction is a technique, which instead is dependent on the background of the scene. The idea behind background subtraction is to utilize a background of the scene, if this can be modelled. Once the background have been modelled, it can be subtracted from each input image, leaving only objects of interest behind.

The background can be modelled by the training data gathered. This is done by averaging all training images of class *table*, resulting in an average background image, as illustrated in 31b.

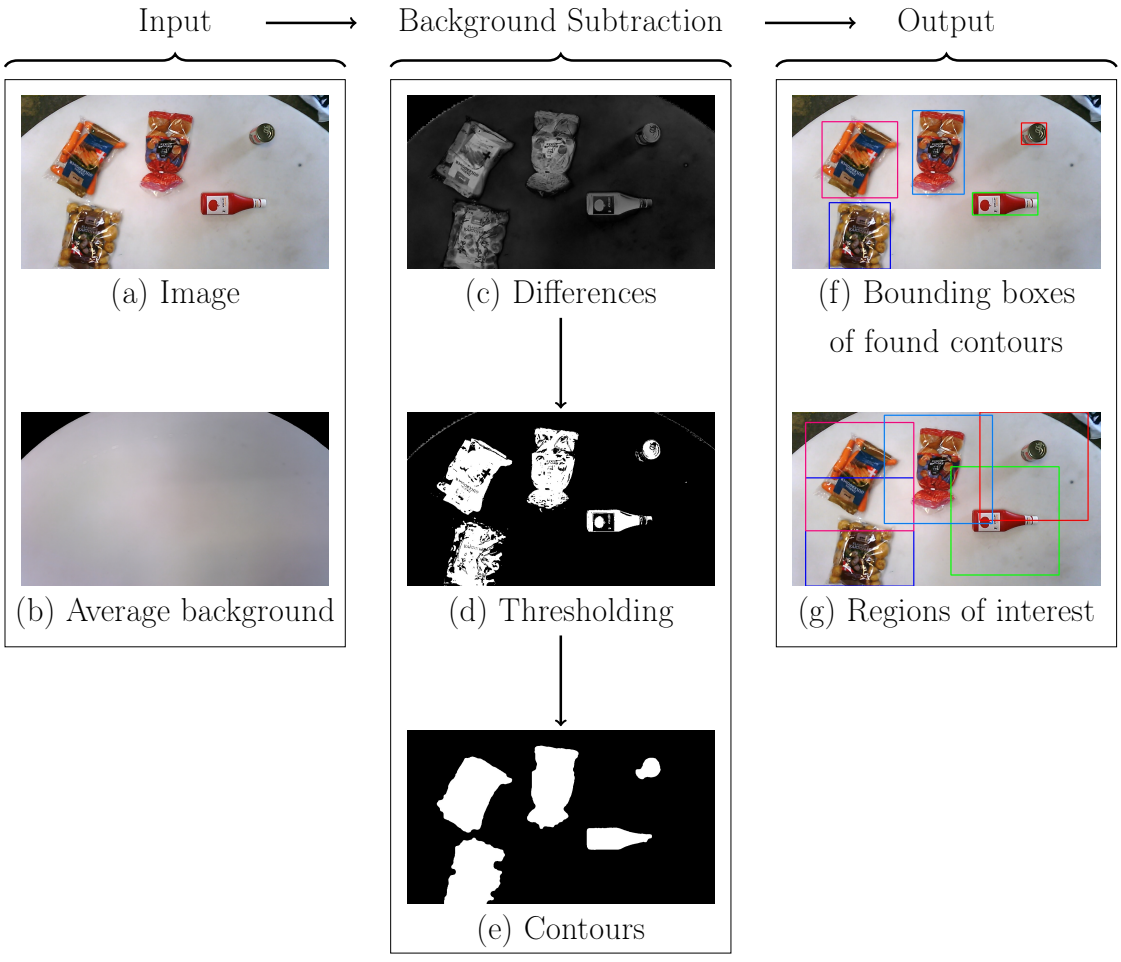
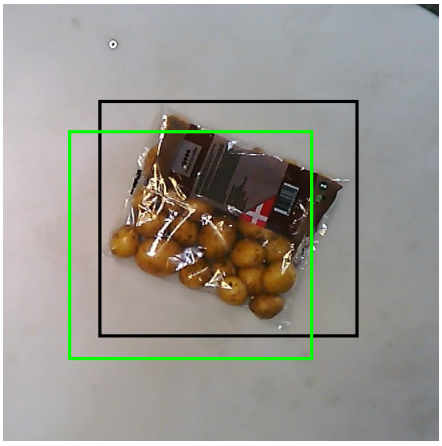


Figure 31: The Process of Background Subtraction

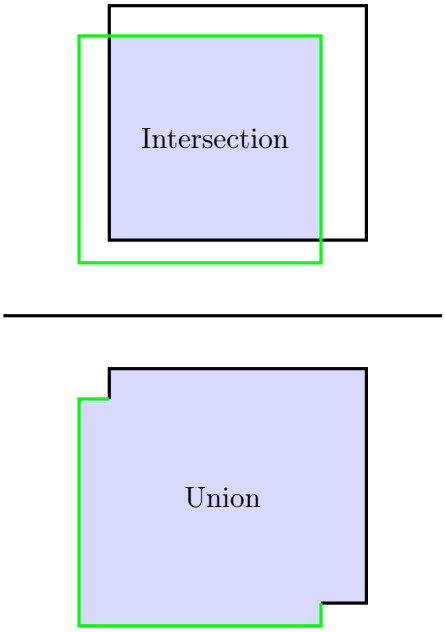
Figure 31 illustrates the process of background subtraction. The average background image shown in figure 31b is subtracted from the input image shown in figure 31a, which yields the difference image illustrated in figure 31c. The difference image is then treated the same way as the probability map in backprojection described in section 6.1.2, which once again results in a bounding box of the objects and a RoI for each found object.

6.2 Evaluating Performance of Localization Methods

In the evaluation and comparison of the above described localization methods, multiple metrics are considered. The first and most important parameter is Intersection over Union, IoU, which is illustrated in figure 32. IoU is a metric used to indicate how well the predicted bounding box is set.



(a) Image with a black ground truth bounding box and a green predicted bounding box



(b) Intersection over union areas

Figure 32: Illustrating Intersection over Union

When evaluating backprojection and background subtraction, it is considered how many of the predicted bounding boxes is acceptable. An IoU score above 0.50 is deemed acceptable. Furthermore, the average IoU of all the classes, except the empty table class, is calculated. The empty table class was discarded because zero regions of interest were detected.

Another metric evaluated is whether the midpoint, of the predicted bounding box, is within the ground truth bounding box. This metric is important, since the naive coordinate to give the robot as a grabbing point is the midpoint of the predicted bounding box.

Method	Average IoU	Above 0.50 IoU [%]	Midpoint within bounding box [%]
Background Subtraction	0.6762	74.65	99.72
Backprojection	0.5786	69.01	96.45

Table 15: Localization test results for all classes performed on the large dataset.

From the results in table 15 the best localization method is background subtraction. The performance of background subtraction, with respect to each of the 8 classes present in the large dataset without the empty table class, can be seen in table 16.

Class	Average IoU	Above 0.50 IoU [%]	Midpoint within bounding box [%]
Carrots	0.8465	100.0	100.0
Ketchup	0.6217	85.15	100.0
Arm	0.7158	96.29	98.31
Bun	0.7875	100.0	100.0
Cat food beef	0.4628	24.91	100.0
Potato	0.8569	100.0	100.0
Cat food salmon	0.4389	12.87	99.66
All classes	0.6762	74.65	99.72

Table 16: Background subtraction IoU test results for each class

It is seen that background subtraction has a hard time predicting an acceptable bounding box for the two cat food classes, while the majority of the predicted bounding boxes for the other classes is acceptable. More test results involving each methods performance can be seen in appendix C.

7 Methods for Object Detection of Consumables

So far classification and localization methods have been explained and tested independently of each other. If classification and localization is combined it would be possible to classify specific objects and localize them. The combination of localization and classification is called object detection. In this section a traditional object detection method will be introduced before moving into combining a localization method with a classification method. Afterwards, neural networks targeted at object detection will be presented.

7.1 Template Matching

Template matching is a simple object detection method. When template matching is performed the algorithm searches for one or several templates in an image.

When template matching is used it slides a template across an image. While the template is slided across the image it generates a matching space by applying a matching metric. The used matching metric is normalized cross correlation, defined in Dawson-Howe (2014) by equation 23.

$$M(i,\ j) = \frac{\sum_{m,n} f(i+m, j+n) \cdot t(m,n)}{\sqrt{\sum_{m,n} f(i+m, j+n)^2 \sum_{m,n} t(m,n)^2}} \tag{23}$$

Where $f(i,j)$ is the i'th row and j'th column pixel value of the input image, and $t(m,n)$ is the m'th row and n'th column pixel value of the template. Instead of applying template matching directly on the input image it is first used in a downscaled version of the input image to find a RoI, which is illustrated in figure 33. This is done to minimize the computations.

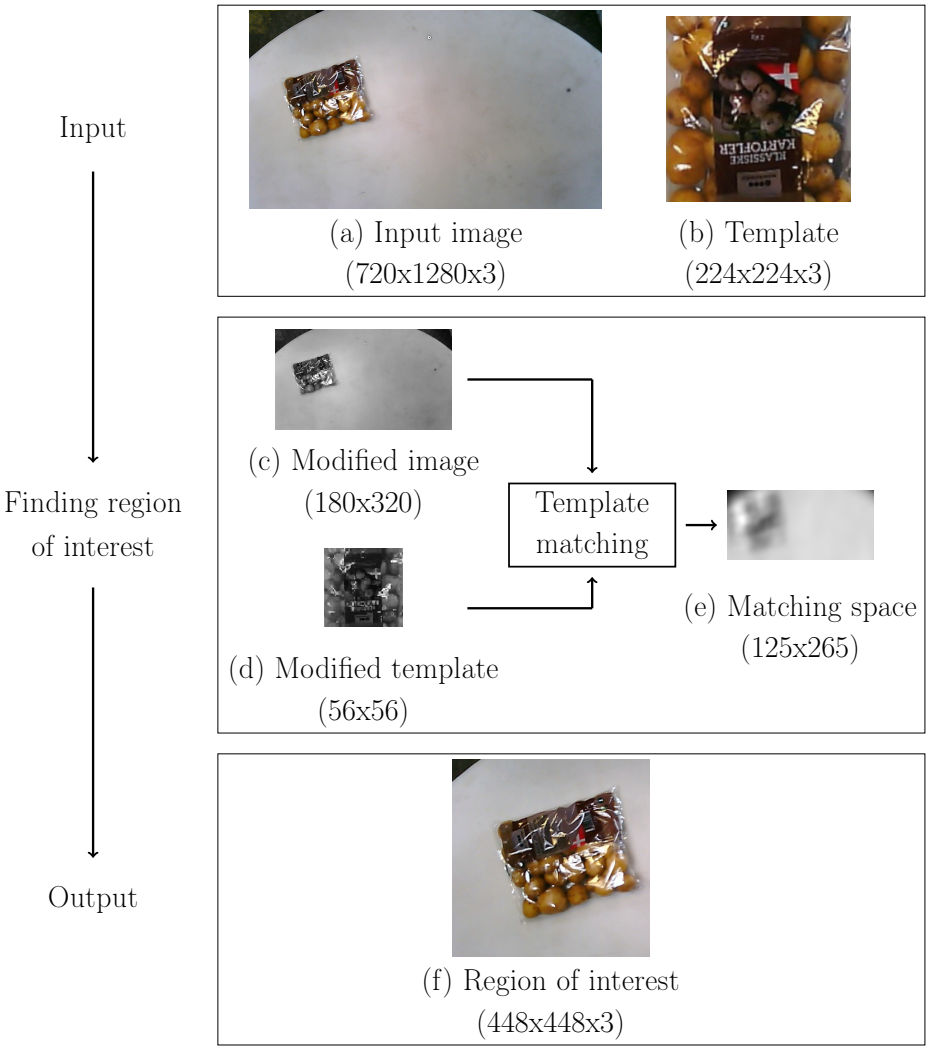


Figure 33: The process of finding regions of interest with template matching

The RoIs are found using a resized version of the input image and of the template. Furthermore, the input image and the template is converted into grayscale images. Once both images have been converted template matching is performed, and the matching space is calculated. The center of the RoI is the center of the template which generates the highest value in the matching space. From this center a region is found by spatially expanding an area to a size of 448x448, as seen in figure 33f.

Once a RoI is found, template matching is then performed once again on the RoI. All of the above is done for each template representing each class, as illustrated in figure 34.

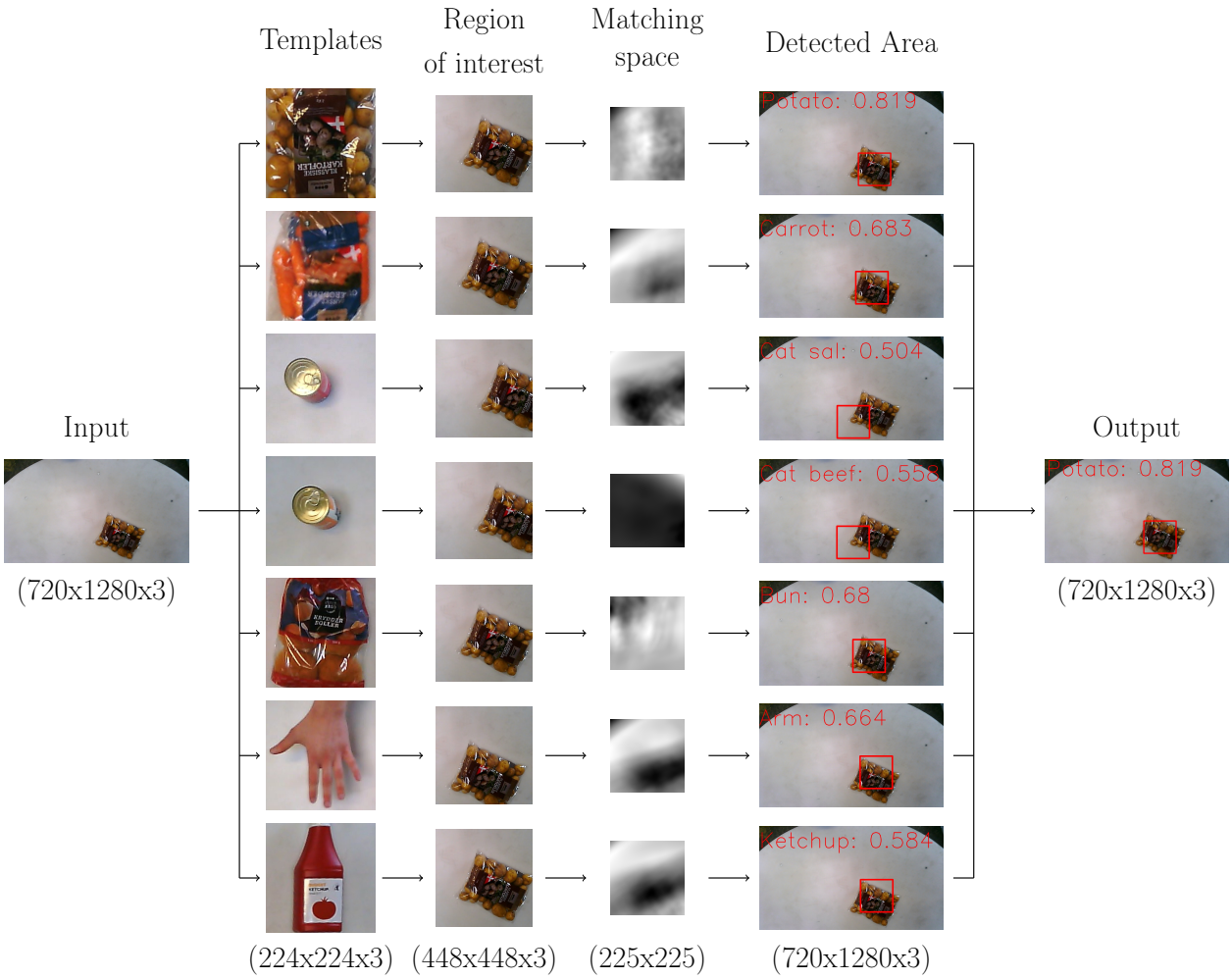


Figure 34: The workflow of template matching

Once matching spaces have been calculated on all the regions of interest the maximum value of each matching space is found. The template resulting in the highest matching value is then applied as the class of the localized object. The bounding box of the detected object is the same size as the template's spatial size, and has the location of the highest matching value.

7.2 Combining Classification and Localization

In section 6.1 two localization techniques for region proposal were presented. Based on the results in table 15 background subtraction is chosen to be used as the regional proposal technique, when combined with classification. Background subtraction can be combined with a classification method in two different ways.

1. The first method is based on using the methods independently of each other: using classification to classify an object in the image and localization to localize it.

2. The second method is based on using the methods in sequence: first localizing all regions of interest and then classifying them.

The second method is favorable, since this allows multiple objects to be present in the scene, while the first method relies on there only being one object present in the image.

The workflow of combining regional proposal and a neural network in sequence is illustrated in figure 35.

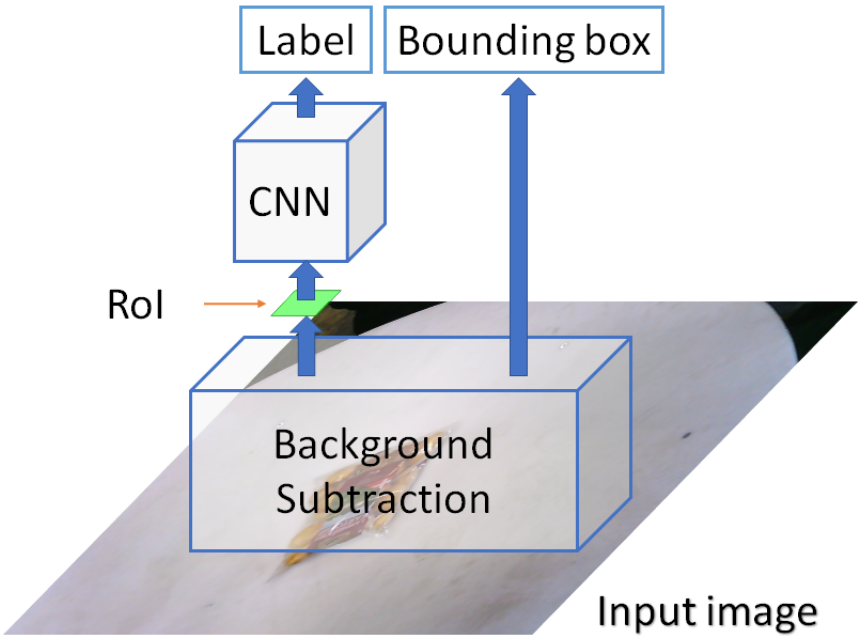


Figure 35: Overview of the combination of region proposal and neural network for localization and classification for an image containing a single object.

As seen in the overview of the method in figure 35 the image is first given to the region proposal technique, which generates a RoI that the neural network then predicts on. The contour, found by the region proposal technique, is used for generating a bounding box.

The neural networks described in section 5 were trained on full images, while this problem is based on classifying objects in RoIs in the image. Therefore, it is necessary to train the neural networks from scratch on the RoIs. The same hyperparameters were used for retraining on the subimages as in the neural networks used for the full images.

To gather training data for the neural network the dataset described in section 3 is used as origin images. Background subtraction is then used on the whole dataset to transform the dataset into subimages. This is done in practice by using the region proposal technique on each image in the dataset to generate a new image that is then put in a new dataset.

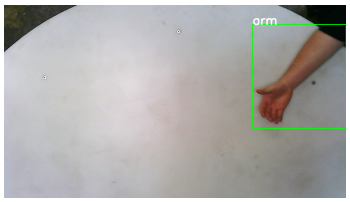
The classification methods is trained on the subimages yeilding the following results.

Summary	Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
kNN	-	-	-	-	0.7730
kNN with thresholding	-	-	-	-	0.9560
FCN	128	0.4330	0.8489	0.4079	0.8614
2-Block CNN	94	0.3100	0.8604	0.1734	0.9682
3-Block CNN	180	0.0883	0.9763	0.0220	0.9955

Table 17: Results from training the networks on the subimages derived from the large dataset.

Based on the results in table 17 the 3-Block CNN is the best classification method, when classifying the subimages images.

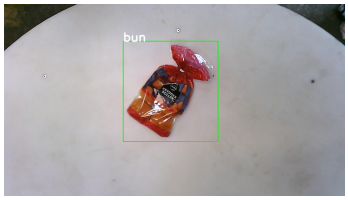
An example of the combined localization and classification is shown in figure 36.



(a) Arm

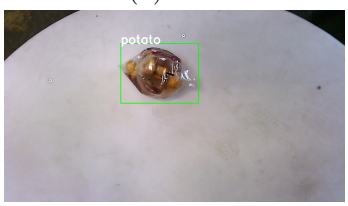
(b) Cat food beef

(c) Carrot

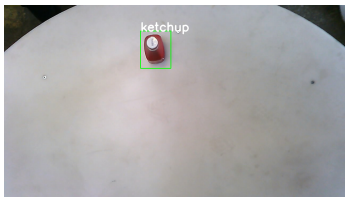


(d) Bun

(e) Cat food salmon



(f) Potato



(g) Ketchup

Figure 36: Examples of objects being located and classified by a combination of background subtraction and a 3-Block CNN.

A noteworthy addition when combining the techniques in sequence is the ability to classify an image containing multiple different objects as shown figure 37.

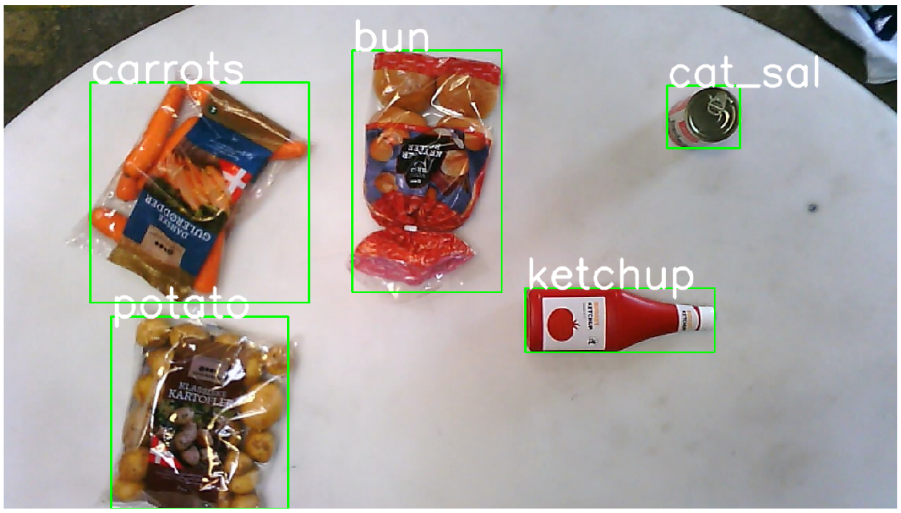


Figure 37: Example of multiple objects being located and classified by a combination of background subtraction and a 3-Block CNN.

The results presented in table 17, shows that the 3-Block CNN achieves an accuracy of 0.9955, when trained on RoIs. This effectively means that the classification error rate has decreased by 33.8% from the classification accuracy shown in section 5.5, table 12, where the 3-Block CNN was trained on full sized images.

7.3 Object Detection with Neural Networks

The last considered object detection method is state of the art neural networks specifically designed for object detection. The implemented neural network for object detection is Faster Region-based Convolutional Neural Network, Faster R-CNN. In order to understand the Faster R-CNN, the earlier networks leading up to Faster R-CNN will be explained, which involves R-CNN and Fast R-CNN.

7.3.1 Region-based Convolutional Neural Network

An object detection neural network can be created by combining a regional proposal method with a CNN. What differentiate this method from the combination of background subtraction and a CNN, is that this method learns to predict the bounding boxes. The R-CNN is presented in paper Girshick et al. (2014). An overview of the R-CNN is illustrated in figure 38.

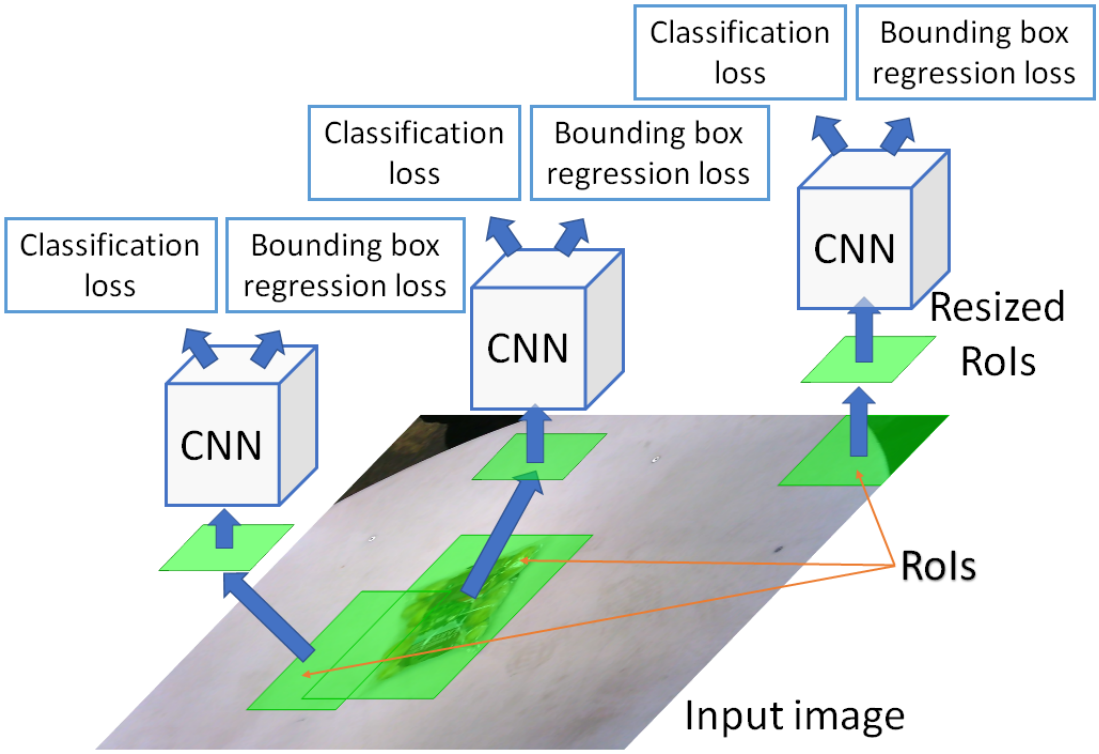


Figure 38: R-CNN system overview

When the R-CNN is detecting objects it generates region proposals, which is then classified by a CNN. The proposed regions are resized into a fixed size required by the CNN. Instead of a single loss being calculated as in the networks designed for classification, the R-CNN calculates a multi regression loss, which consists of a classification loss and a bounding box regression loss. The bounding box regression loss is calculated upon the offset and scaling of the region proposals with respect to the ground truth.

Since the R-CNN operates on each regional proposal, where many are overlapping, without sharing computations, the method becomes inefficient. In the paper Girshick (2015), a R-CNN based upon the VGG16 architecture required 47 seconds of processing per image using a GPU. By sharing the computations of the CNN, the time efficiency would improve. This idea is the foundation for the network Fast R-CNN.

7.3.2 Fast Region-based Convolutional Neural Network

The Fast R-CNN differs from the R-CNN, since the CNN calculates a feature map for the entire image, which drastically reduces the computations of the CNN. The region proposals are generated in the same manner as the R-CNN, and then projected onto the feature map. Fast R-CNN is presented in the paper Girshick (2015). An illustration of the Fast R-CNN

structure is shown in figure 39.

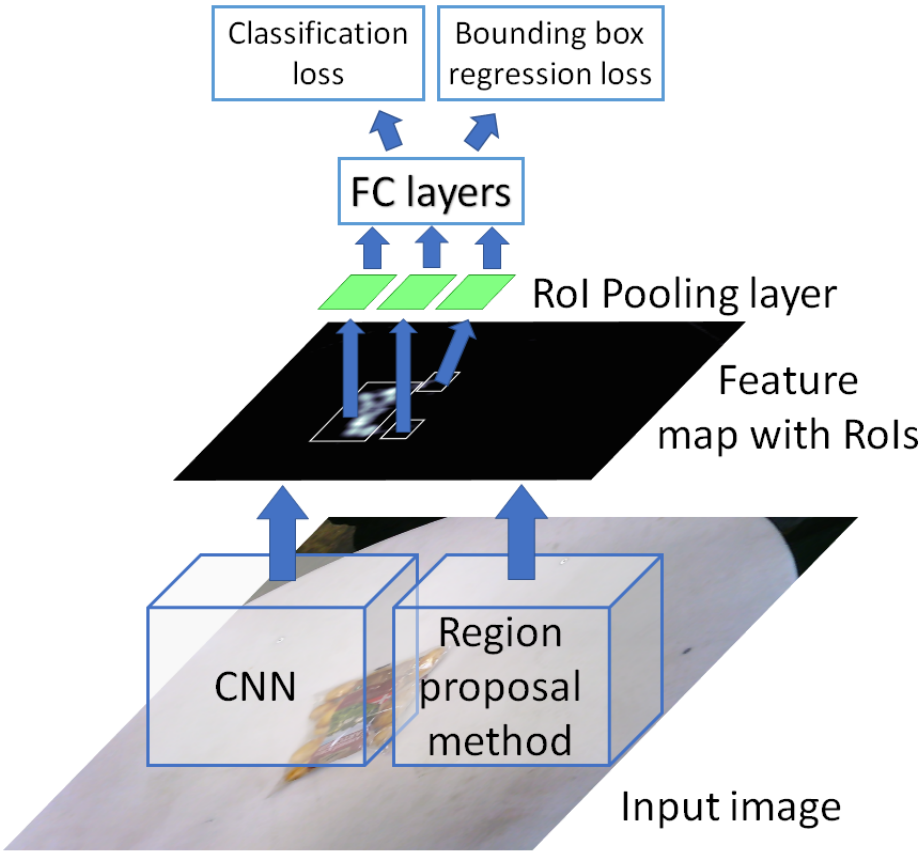


Figure 39: Fast R-CNN system overview

When an input image is given to the Fast R-CNN, it calculates a feature map and projects the proposed regions onto the feature map. Each RoI in the feature map is pooled to fit into the fully connected layer.

The Fast R-CNN produces a multi regression loss, which consists of a classification loss and bounding box regression loss, similar to R-CNN.

The major difference between R-CNN and Fast R-CNN is their performance with respect to time. The time spend on processing an image with the Fast R-CNN is reduced to 2.32 seconds per image on a GPU, as shown in the paper Girshick (2015), which is caused by the shared computations of the Fast R-CNN. The new bottleneck with respect to time is the proposal method according to the paper Ren et al. (2015), which is the problem leading to the rise of Faster R-CNN.

7.3.3 Faster Region-based Convolutional Neural Network

The Faster R-CNN is an optimization of the Fast R-CNN, which utilizes a Regional Proposal Network, RPN, to take advantage of the computations from a feature extractor CNN.

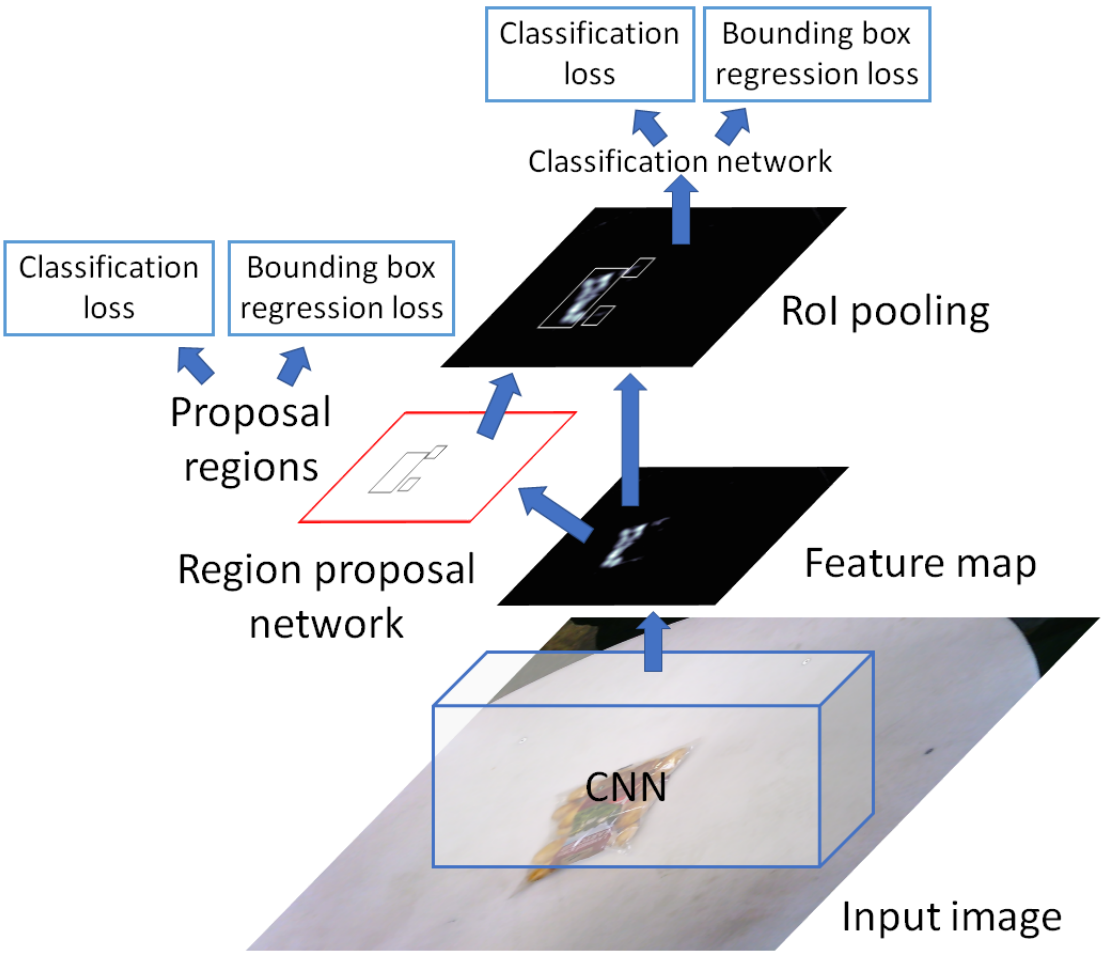


Figure 40: Faster R-CNN system overview

Figure 40 illustrates the structure of a Faster R-CNN. It is seen that a feature extractor CNN is used to generate a feature map. This feature extractor is a pretrained CNN and is not trained during the training phase of the Faster R-CNN. The feature map is fed through a RPN which calculates a set of proposals, with objectness scores. An objectness score represents the degree to which a region is an object versus a background. The output of the RPN is then fed into two different layers, one for objectness classification and another for box regression. The operation of the RPN, at one position in the feature map, is shown in figure 41.

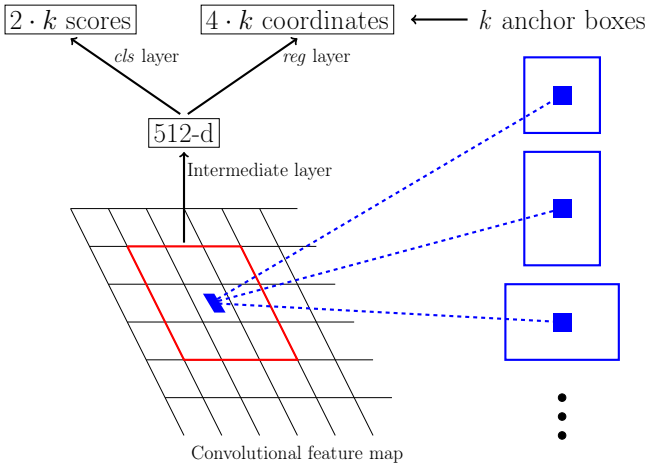


Figure 41: Application of region proposal network at a single position in the feature map.

At each application of the RPN, multiple regions proposals are found, where the number of maximum possible proposals, anchor boxes, for each position is denoted as k. The regression, *reg*, layer have $4 \cdot k$ outputs, representating the coordinates of k anchor boxes. The classification, *cls*, layer have $2 \cdot k$ outputs, which estimates the probability of object or not object for each proposal.

The k anchor boxes is generated from combinations of different scaling and aspect ratios. The bounding box regression loss is calculated for k linear regressors, where there is a linear regressor for each of the combinations of scale and aspect ratio. After the processing of the anchors they are filtered using non-maxima suppresion, this is done based on their overlap and *cls* score. After non-maxima suppresion, the proposals are delivered to the classification network.

In addition to the so far explained, the implemented Faster R-CNN can discard regions after the classification network, based on a threshold of the classification score.

Figure 42 shows the performance of Faster R-CNN with different thresholds and the corresponding validation accuracy. It can be seen that the overall best performance is achieved using a threshold of 0.8, which yields an average IoU of 0.7492, 98.15% of all IoUs above 0.5, 98.84% of localizations with midpoint within ground truth bounding box and a validation accuracy of 0.9809.

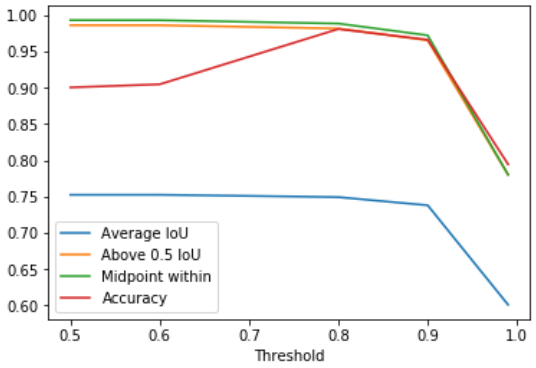


Figure 42: Faster R-CNN with different thresholds for object detection.

7.4 Evaluating Object Detection Methods

In the evaluation of the metrics for object detection, a combination of the metrics from classication and localization is used, to indicate the performance.

Method	Average IoU	Above 0.50 IoU [%]	Midpoint within [%]	Accuracy
Template Matching	0.2425	15.14	53.82	0.1438
Template Matching with multispectral band thresholding	0.4065	32.99	96.31	0.4630
Background subtraction and 3-Block CNN	0.6762	74.65	99.72	0.9955
Faster R-CNN	0.7492	98.15	98.84	0.9809

Table 18: Object detection test results for all classes.

It is from the results in table 18 seen that the Faster R-CNN by far achieves the highest IoU scores, indicating that this method is superior, when it comes to the accuracy of the bounding box. Furthermore, the combination of background subtraction and the 3-Block CNN achieves the highest midpoint within, closely followed by Faster R-CNN. However, looking at the classification validation accuracy, the combination of background subtraction and the 3-Block CNN achieves the highest score.

The methods are further compared by measuring their prediction time and memory usage.

Summary	Prediction time [ms]	Memory usage [MB]
Template Matching	4013.1	0.155
Template Matching with thresholding	4136.0	0.155
Background subtraction and 3-Block CNN	43.4	33
Faster R-CNN	971.6	522

Table 19: Time used for a single prediction meaned over 30 predictions and memory usage for the object detection methods.

It is seen in table 19 that the computational time of the 3-Block CNN combined with

background subtraction is much lower than the other methods. Furthermore, the Faster R-CNN has a much higher memory requirement than the other methods.

8 Discussion of Methods and Future Work

This section will contain discussions of the presented methods throughout the report and will also discuss relevant future work.

Indirect Overfitting Caused by Limited Data

In section 3 it was chosen to only split the data in two splits: training and validation. This could introduce a problem when evaluating the designed neural networks. Currently the validation data is being used for optimization of hyperparameters, and as a performance measure of how the neural networks performs on unseen data. When hyperparamter optimization is performed on the validation data, there is a risk of the network becoming indirectly overfit of the validation data through the choice of hyperparameters. If that is the case, then it becomes a bad indication of performance on unseen data. Therefore, the amount of data should be expanded, making it possible to split the data in 3 splits: training, validation and test.

Zero Padding before Convolutional Layers

In section 4.7, figure 10, it is shown, that the input is downsampled, when the convolutional filter is applied. This is caused by the filter not being applicable at the borders. By applying zero padding prior to the convolutional layer the unintelligent downsampling, introduced by the filter not being applicable at the border, could be avoided. The designed CNNs in section 5 is only respectively 2 and 3 blocks deep, therefore the unintelligent downsampling might not be a problem, but if the CNNs were to be made deeper the effects of not zero padding would become greater.

Trainable Activation Function Instead of ReLU

In section 4.3 it was chosen to use the ReLU activation function between layers in the designed neural networks. The paper Ramachandran et al. (2017) introduces Swish, a learn able activation function. It is shown in Ramachandran et al. (2017), that Swish consistently outperforms ReLU in the investigated cases. Based on this, it should be investigated if the usage of Swish improves the designed neural networks.

Creating More Data with Domain Randomization

Domain randomization described in paper Tobin et al. (2017), is used as a technique to train a neural network in a simulation for usage in the real world. This is done by

randomly rendering the objects in the scene and training on the renderings. Using the ideas behind domain randomization, the objects of interest in this thesis could in the same manner be rendered in different scenes to make the neural network robust to changes in the environment.

Optimization of Hyperparameters

Finding the optimal parameters of a method is crucial to obtaining a high performance, thus hyperparameter optimization is important. In the section 5.4 the values of the parameters were chosen manually for each method based on experience, thereby performing grid search. Instead of using the rough estimates from grid search directly, a finer search in the region near these could have been performed using random search. This could result in more optimal hyperparameters.

As described in section 7.2 the same hyperparameters were re-used when training the networks on the subimages generated from background subtraction. Because the neural networks were trained on a new dataset, it might have shown to be favorable to perform hyperparameter optimization on the new dataset, since the hyperparameters from the full images is not necessarily the optimal.

Transfer Learning for Classification

The principle of transfer learning is to transfer the *experience* learned of one neural network, into a new or modified version of the neural network to be used on a similar problem. The motivation behind transfer learning is that training a big and complex neural network from scratch would be very time consuming and would require lots of data. Since data is somehow limited in this bachelor thesis, it is worth considering transfer learning. Unfortunately, the project group was not able to develop a method for transfer learning within the time frame of the project. If the problem was to be expanded or the difficulty of the problem rises in complexity, it could be relevant to explore the possibilities of transfer learning.

Dynamic Background Modelling for Background Subtraction

The background modelling described in section 6.1.3 is vulnerable to changes in the environment, which might be a problem if this method is to be implemented. The changes in environment includes: changes in luminance, miscolor on the work table and similar. To avoid the effects of environment changes similar to the mentioned, a dynamic background

modelling could be performed.

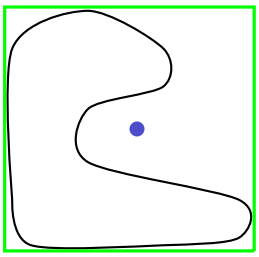
To perform dynamic background modelling the training data is averaged to generate an initial background model. After this the background model is updated with each image taken by the system. The update rule is illustrated in equation 24.

$$b_{t+1}(i,j) = \alpha\ x_t(i,j) + (1-\alpha)\ b_t(i,j) \tag{24}$$

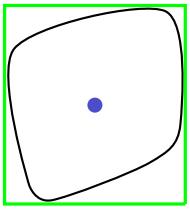
where b is the background model and x is the input image. The update rate α is to be determined based on the frequency of input images and how fast the background model should incorporate changes.

Robot Grasping Point

The thesis only focuses on localization of the objects with a bounding box, where the grasping point of the robot is currently defined as the center of the bounding box. The center of the bounding box might not be a suitable grasping point, since some of the objects are located in bags and can therefor resemble abstract shapes, when seen from above. This is illustrated in figure 43.



(a) Midpoint of bounding box not being a suitable grasping point



(b) Midpoint of bounding box as an acceptable grasping point

Figure 43: Illustration of midpoint as grasping point. Objects is represented by a shape outlined with a black line, bounding box with green line and midpoint of bounding box with a blue circle.

The example shows, that the midpoint of the bounding box can be located outside the object. A way to avoid using the midpoint as grasping point, could be to calculate a probability map, like backprojection in section 6.1.2, and calculating a weighted center. Another solution to this problem, could be to use a neural network for segmentation, for example Mask R-CNN, which can be used to predict an object mask. A suitable grasping point can then be found from this object mask.

9 Conclusion

During the thesis methods for classification, localization and object detection was explored.

Data were collected in two rounds because the first dataset was too small to evaluate the developed methods. The final dataset used consists of 2347 images distributed among 8 classes. When collecting the data, the acceptable resolution of the camera was found to be 720x1280. Furthermore, a data structure consisting of a training split and a validation split with a distribution ratio of 4:1 was used to ensure the methods were trained, optimized and evaluated correctly compared to the total size of the dataset. Utilizing image mean subtraction and normalization on the data, the data was preprocessed properly for the neural networks.

Using the high level neural network API, Keras, multiple neural networks were designed, with the purpose of classifying images by applying a well defined label. Using a block structure consisting of a convolutional layer with ReLU activation function followed by a max pooling layer with pooling size of 2x2 and stride of 2, it was discovered that a convolutional neural network consisting of 3 blocks was the best solution of classification by achieving an accuracy of 99.32%.

Two localization methods were designed, where the best localization method was shown to be background subtraction, which achieved an Intersection over Union, IoU, score above 0.50 in 74.65 of all cases.

By retraining the convolutional neural network on images provided by background subtraction, it is possible to reach a classification accuracy of 99.55%. Furthermore, by combination of the background subtraction and the convolutional network consisting of 3 blocks, a simple object detection network was created, which showed to have a prediction time of 43 milliseconds.

The designed object detection method was compared with Faster Regional-based Convolutional Neural Network, Faster R-CNN, which is a state of the art neural network targeted at object detection. This showed to achieve a much higher localization performance with 98.15% of all localization having an IoU above 0.5. However, Faster R-CNN achieved a lower classification accuracy found to be 98.09% with a prediction time of 971 milliseconds.

The methods based upon usage of neural networks showed to outperform traditional computer vision techniques, that did not contain a neural network in both classification and object detection. For classification it showed that using the convolutional neural net-

work build from 3 blocks achieved a 84.5% lower classification error rate than k-Nearest Neighbor. For object detection it showed that the combination of the 3 block convolutional neural network achieved a 99.16% lower accuracy error rate and 42 percentage point higher amount of IoUs above 0.5 compared with Template Matching.

10 Bibliography

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV*. Wiley Publishing, 1st edition.

Devasena, C. L., Sumathi, T., Gomathi, V., and Hemalatha, M. (2011). Effectiveness evaluation of rule based classifiers for the classification of iris data set. *Bonfring International Journal of Man Machine Interface*, 1(Special Issue Inaugural Special Issue):05–09.

Dunne, R. A. and Campbell, N. A. (1997). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer.

ELP, C. M. F. S. (2013). Sony imx214 webcam. `https://www.aliexpress.com/item/13-Megapixel-Autofocus-USB-Camera-High-Resolution-USB2-0-SONY-IMX214-Color-CMOS-Mini-Webcam-Camera/32909008612.html?fbclid=IwAR2Xkw5758lqk2K-LyN4otorUFYzV6witqcCv09OpLgGKhtF_joL4774Byc`. Accessed on 25/05-2019.

Girshick, R. (2015). Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Glorot, X. and Bengio, Y. (2018). Understanding the difficulty of training deep feedforward neural networks. 2010. *Received February*, 12.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward

networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

Appendices

The attached appendices contains relevant information to support the report, with additional results than those shown through the report and with in depth explanation of selected parts.

A Public Available Code

The code developed through the project have been made publicly available at the project's Github repository: `https://github.com/ancker1/BSc-PRO`.

The implementation of Faster R-CNN is based on the following Github project: `https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras`.

B Structure of Small Dataset

The structure of the first and small dataset, which showed too small to evaluate the methods properly, is shown in figure 44.

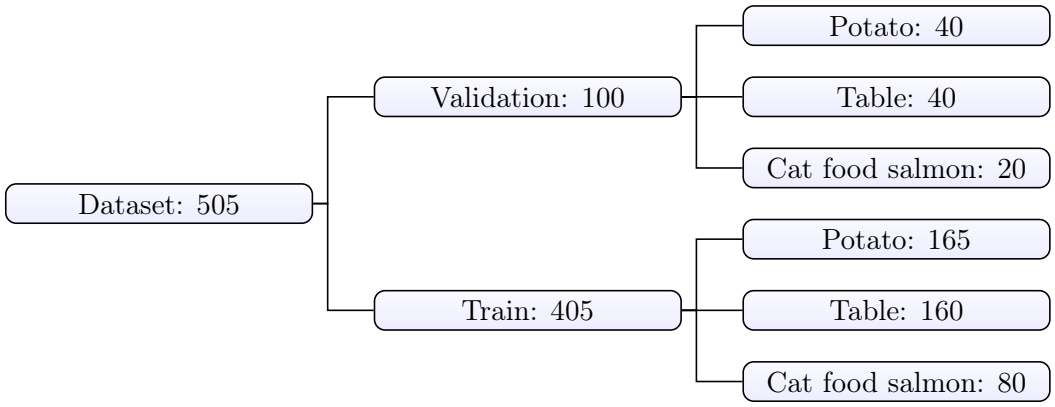


Figure 44: Structure of the small dataset with 3 classes.

C IoU Test Results

Class	Average IoU	Above 0.50 IoU [%]	Midpoint within bounding box [%]
Carrots	0.7047	93.15	100.0
Ketchup	0.6384	79.69	100.0
Arm	0.4853	47.47	99.32
Bun	0.8260	99.40	100.0
Cat food beef	0.4025	38.56	92.49
Potato	0.7134	98.98	100.0
Cat food salmon	0.2488	21.78	82.83
All classes	0.5786	69.01	96.45

Table 20: Backprojection IoU test results for each class

Class	Average IoU	Above 0.50 IoU [%]	Midpoint within bounding box [%]
Carrots	0.4376	35.61	91.09
Ketchup	0.0920	04.54	20.30
Arm	0.2670	14.14	67.00
Bun	0.4144	17.06	91.61
Cat food beef	0.0345	00.00	06.14
Potato	0.4290	36.02	91.24
Cat food salmon	0.0233	00.00	05.94
All classes	0.2425	15.14	53.82

Table 21: Template matching without multispectral band thresholding IoU test results for each class

Class	Average IoU	Above 0.50 IoU [%]	Midpoint within bounding box [%]
Carrots	0.5425	60.95	100.0
Ketchup	0.5156	49.69	97.57
Arm	0.3665	15.82	97.64
Bun	0.5104	48.50	100.0
Cat food beef	0.1907	00.00	91.46
Potato	0.5240	52.86	100.0
Cat food salmon	0.1745	00.00	87.12
All classes	0.4065	32.99	96.31

Table 22: Template matching with multispectral band tresholding IoU test results for each class

D Hyperparameter Optimization Results for Classification

In this section the results for hyperparameters of every classification method are summarized in tables below. The training- and validation- accuracies shown in the tables are the best achieved accuracies of the respective models and the tested hyperparameters. Furthermore loss and accuracy plot of both training and validation is illustrated of the models to ensure overfitting is not present. The plot is only of the model which achieved the highest validation accuracy. The different hypeparameters were tested for all possible

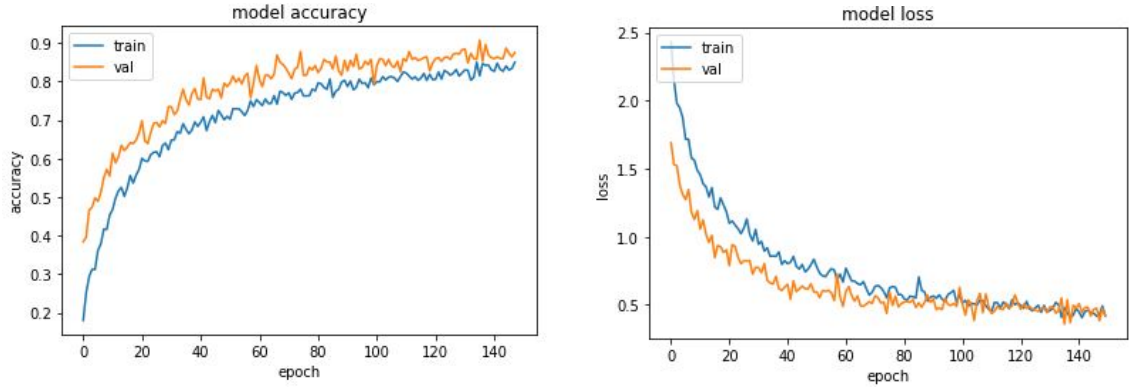
combinations.

D.1 Fully Connected Network

The FCN network have two hyperparameters to optimize learning rate and hidden size. A learning rate $1e-3$ and $1e-4$ were tested and a hidden size $32, 64, 128, 256$ and 512 were tested. The result is shown in table 23.

Hyperparameter Fully Connected Network (FCN) Results			
Learning Rate	**Hidden size**	**Training Accuracy**	**Validation Accuracy**
1e-3	32	0.302767	0.659091
1e-3	64	0.161150	0.320455
1e-3	128	0.147043	0.245455
1e-3	256	0.145958	0.293182
1e-3	512	0.142702	0.204545
1e-4	32	0.837765	0.904545
1e-4	64	0.849702	0.906818
1e-4	128	0.855128	0.890909
1e-4	256	0.868150	0.881818
1e-4	512	0.875746	0.888636

Table 23: Hyperparameter optimization for FCN runned for 150 epoch per model



(a) Plot of accuracy versus epochs

(b) Plot of loss versus epochs

Figure 45: Plot of accuracy and loss of FCN with Learning rate: $1e-4$ and Hidden size: 64

Based on the results in table 23 the optimal hyperparameters is chosen to be, learning rate $= 1e-4$ and hidden size $= 64$. The hidden size is a balance between having enough weights to classify the different classes but not too many to adjust. In figure 46 shows correlation between validation accuracy and hidden size with a learning rate of $1e-4$. From the figure it can be seen a lower hidden size leads to a better validation accuracy. Furthermore from a plot of the best hyperparameters in figure 45 it is shown overfitting is not present.

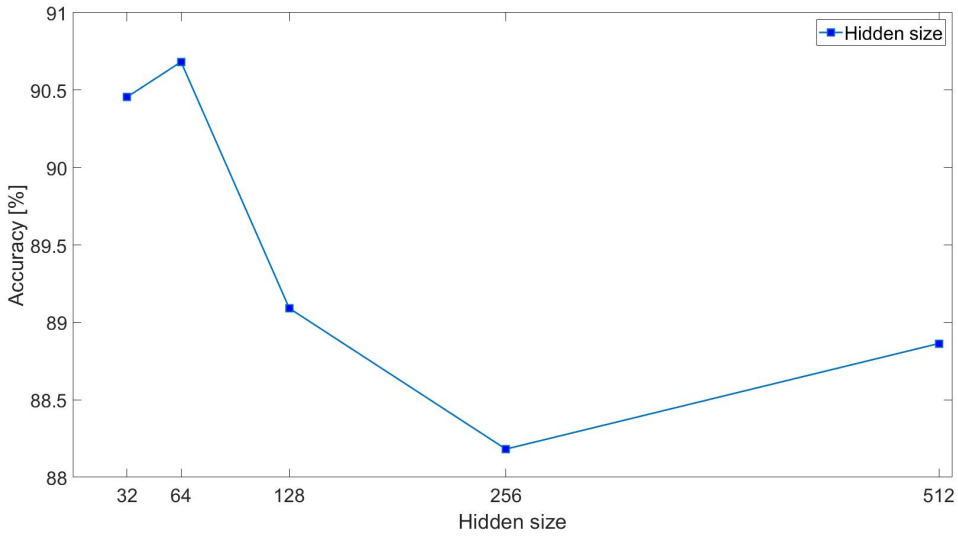


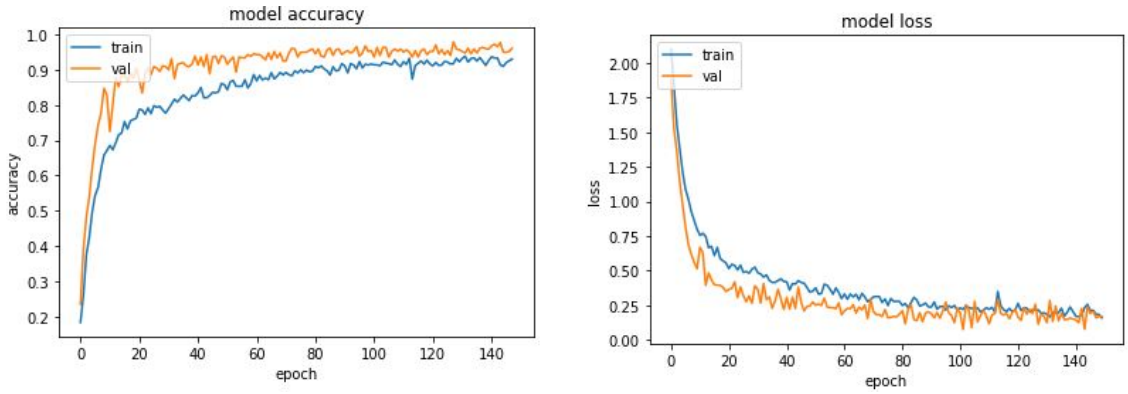
Figure 46: Validation accuracy versus Hidden size with Learning rate $= 1e-4$

D.2 2-Block Convolutional Neural Network

The 2-Block CNN is optimized for 3 hyperparameters.

Hyperparameter Optimization 2-Block CNN				
Learning Rate	**Conv Size**	**Pool Size**	**Best Training Accuracy**	**Best Validation Accuracy**
1e-2	3	2	0.151926	0.172727
1e-2	3	4	0.639175	0.734091
1e-2	5	2	0.151384	0.170445
1e-2	5	4	0.151926	0.172727
1e-3	3	2	0.938687	0.979545
1e-3	3	4	0.916983	0.963636
1e-3	5	2	0.937602	0.970455
1e-3	5	4	0.935431	0.961364
1e-4	3	2	0.824200	0.945455
1e-4	3	4	0.706999	0.775000
1e-4	5	2	0.730874	0.940909
1e-4	5	4	0.762887	0.872727

Table 24: Hyperparameter optimization of 2-Block CNN, 150 epoch pr. model on all 8 classes



(a) Accuracy versus Epoch

(b) Loss versus Epoch

Figure 47: Optimal hyperparameter plot of 2-Block CNN with learning rate 1e-3, convolutional filter size 3x3 and pooling size 2x2

Based on the results in table 24 the optimal hyperparameters is chosen to be, learning rate $= 1e-3$, convolution size 3×3 and pool size 2×2. Furthermore overfitting is not present which can be seen from the plot of the hyperparameters in figure 47.

E Finding Human Classification Accuracy

The purpose of finding human accuracy is to see if a given classification method is better than a human at classifying on the large dataset.

The project group went through the whole validation split in the large dataset labelling all images. If the project group gave an image a wrong label compared to the ground truth it counted as an error. Furthermore if the project group were in doubt or disagreed it counted as an error.

The total amount of images in the validation split in the large dataset is 472. The total amount of errors and the distribution of the errors is shown in table 25.

Class	Errors
Arm	0
Potato	0
Ketchup	0
Table	0
Cat food beef	10
Cat food salmon	8
Carrots	0
Bun	1
Total amount of errors	19

Table 25: Results from classification on the validation split of the large dataset by the project group.

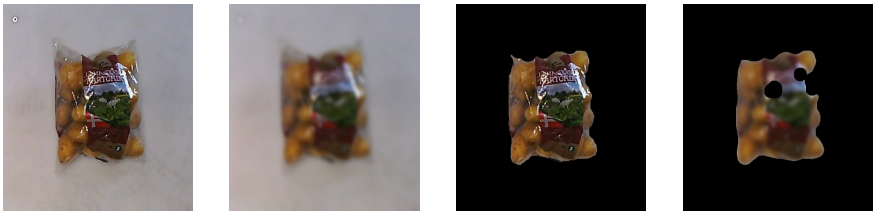
It is seen that the total amount of errors is 19. This means that the human accuracy is 0.9597.

F Hyperparameter Optimization of k-Nearest Neighbor

The k-Nearest Neighbor classification method is tested with different parameters. The pixel feature is shown in table 26 and 27 and the histogram feature is shown in table 28 and 29. Furthermore, the method is tested on full images, shown in table 26 and 28, and cropped images, shown in table 27 and 29.

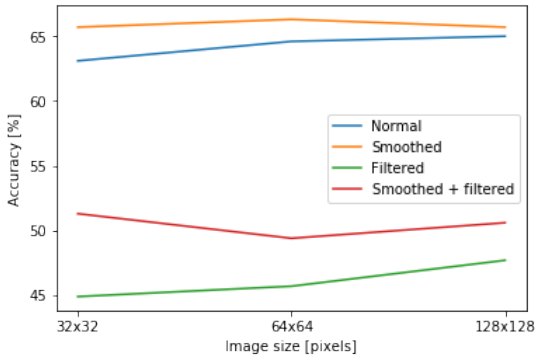
The training data, for the k-Nearest Neighbor method, is preprocessed with different methods. In the following tables the images which are preprocessed with a Gaussian filter is called smoothed. The second preprocessing method is to filter the image with multiband thresholding, this preprocessing method is referred to as filtered in the following tables. The last preprocessing method is a combination of the two previous methods and is referred to as smoothed and filtered in the following tables. A sample of the different types of training data is illustrated in figure 48.

(a) Full (b) Full (smoothed) (c) Full (filtered) (d) Full (smoothed and filtered)

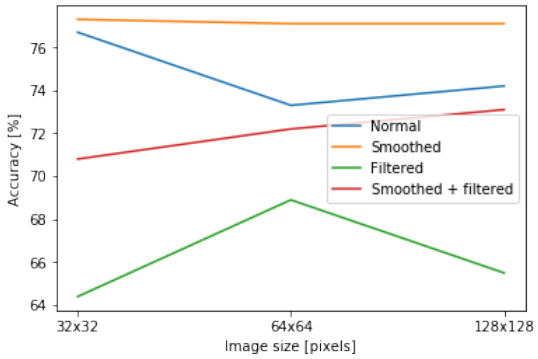


(e) Cropped (f) Cropped (smoothed) (g) Cropped (filtered) (h) Cropped (smoothed and filtered)

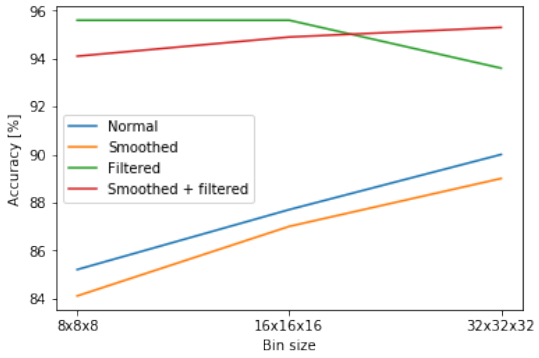
Figure 48: Illustration of the different preprocessing methods for kNN.



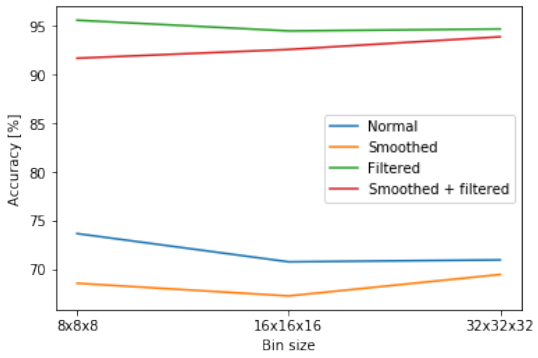
(a) k-Nearest Neighbor with pixel feature on full images.



(b) k-Nearest Neighbor with pixel feature on subimages.



(c) k-Nearest Neighbor with histogram feature on full images.



(d) k-Nearest Neighbor with histogram feature on subimages.

Figure 49: Plots of results from different k-Nearest Neighbor methods for subimages and full images.

The following tables show the results of the hyperparameter optimization for k-Nearest Neighbor classification method.

Pixel feature on full images				
Image type	**Image size**	**Memory usage**	**Prediction time**	**Accuracy**
Normal	32x32	46.08 [MB]	0.0605 [s]	63.1 %
	64x64	184.32 [MB]	0.241 [s]	64.6 %
	128x128	737.28 [MB]	0.928 [s]	65.0 %
Smoothed	32x32	46.08 [MB]	0.0578 [s]	65.7 %
	64x64	184.32 [MB]	0.207 [s]	66.3 %
	128x128	737.28 [MB]	0.784 [s]	65.7 %
Filtered	32x32	46.08 [MB]	0.0616 [s]	44.9 %
	64x64	184.32 [MB]	0.235 [s]	45.7 %
	128x128	737.28 [MB]	0.859 [s]	47.7 %
Smoothed and filtered	32x32	46.08 [MB]	0.0774 [s]	51.3 %
	64x64	184.32 [MB]	0.211 [s]	49.4 %
	128x128	737.28 [MB]	0.859 [s]	50.6 %

Table 26: Results of pixel feature on full images

Pixel feature on cropped images				
Image type	**Image size**	**Memory usage**	**Prediction time**	**Accuracy**
Normal	32x32	46.08 [MB]	0.0588 [s]	76.7 %
	64x64	184.32 [MB]	0.234 [s]	73.3 %
	128x128	737.28 [MB]	0.935 [s]	74.2 %
Smoothed	32x32	46.08 [MB]	0.0661 [s]	77.3 %
	64x64	184.32 [MB]	0.210 [s]	77.1 %
	128x128	737.28 [MB]	0.855 [s]	77.1 %
Filtered	32x32	46.08 [MB]	0.0647 [s]	64.4 %
	64x64	184.32 [MB]	0.230 [s]	68.9 %
	128x128	737.28 [MB]	0.826 [s]	65.5 %
Smoothed and filtered	32x32	46.08 [MB]	0.0586 [s]	70.8 %
	64x64	184.32 [MB]	0.212 [s]	72.2 %
	128x128	737.28 [MB]	0.817 [s]	73.1 %

Table 27: Results of pixel feature on subimages

Histogram feature on full images				
Image type	**Bin size**	**Memory usage**	**Prediction time**	**Accuracy**
Normal	8x8x8	7.68 [MB]	0.0207 [s]	85.2 %
	16x16x16	61.44 [MB]	0.0954 [s]	87.7 %
	32x32x32	491.52 [MB]	0.569 [s]	90.0 %
Smoothed	8x8x8	7.68 [MB]	0.0141 [s]	84.1 %
	16x16x16	61.44 [MB]	0.0769 [s]	87.0 %
	32x32x32	491.52 [MB]	0.520 [s]	89.0 %
Filtered	8x8x8	7.68 [MB]	0.0238 [s]	95.6 %
	16x16x16	61.44 [MB]	0.113 [s]	95.6 %
	32x32x32	491.52 [MB]	0.823 [s]	93.6 %
Smoothed and filtered	8x8x8	7.68 [MB]	0.0165 [s]	94.1 %
	16x16x16	61.44 [MB]	0.0817 [s]	94.9 %
	32x32x32	491.52 [MB]	0.524 [s]	95.3 %

Table 28: Results of histogram feature on full images

Histogram feature on cropped images				
Image type	**Bin size**	**Memory usage**	**Prediction time**	**Accuracy**
Normal	8x8x8	7.68 [MB]	0.0148 [s]	73.7 %
	16x16x16	61.44 [MB]	0.0822 [s]	70.8 %
	32x32x32	491.52 [MB]	0.572 [s]	71 %
Smoothed	8x8x8	7.68 [MB]	0.0147 [s]	68.6 %
	16x16x16	61.44 [MB]	0.0784 [s]	67.3 %
	32x32x32	491.52 [MB]	0.563 [s]	69.5 %
Filtered	8x8x8	7.68 [MB]	0.0244 [s]	95.6 %
	16x16x16	61.44 [MB]	0.0804 [s]	94.5 %
	32x32x32	491.52 [MB]	0.572 [s]	94.7 %
Smoothed and filtered	8x8x8	7.68 [MB]	0.0151 [s]	91.7 %
	16x16x16	61.44 [MB]	0.0784 [s]	92.6 %
	32x32x32	491.52 [MB]	0.546 [s]	93.9 %

Table 29: Results of histogram feature on subimages

The memory usage of the whole trainingdata is calculated as follows:

$$\frac{Width \cdot Height \cdot Depth \cdot Format \cdot Total\ images}{1000000} = Mega\ Bytes$$

The width, height and depth are the dimensions of the each training data. The format is the computer number format, for example if *float64* is used the number of bytes is eight. Then, the memory usage is multiplied with the total number of images. Thereby, the amount of bytes used in the whole trainingdata is found. To convert to mega bytes the number of bytes is divided with a million. For example with a image of the size 32x32x3:

$$\frac{32 \cdot 32 \cdot 3 \cdot 8 \cdot 1875}{1000000} = 46.08\ [MB]$$

F.1 k-Nearest Neighbor without Cat Food Beef

The k-Nearest Neighbor classification method is tested without the object cat food beef, which is removed from the large dataset. From appendix F the best methods, for the k-Nearest Neighbor method, was histogram feature on normal images and histogram feature on filtered images. Therefore, only these two methods will be run without the object cat food beef. The results can be seen in table 30.

Histogram feature on full images				
Image type	**Bin size**	**Memory usage**	**Prediction time**	**Accuracy**
Normal	32x32x32	430.18 [MB]	0.452 [s]	92.5 %
Filtered	8x8x8	6.72 [MB]	0.0120 [s]	98.5 %

Table 30: Results of k-Nearest Neighbor without Cat Food Beef

G Derivative of Softmax and Cross Entropy Loss

First the derivate of the softmax function is found, afterwards the derivative of the softmax function is combined with the cross entropy loss to create a simple derivative of both.

G.1 Deriving Softmax

The softmax function is defined by:

$$\sigma(\boldsymbol{s}_i) = \frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}} \tag{25}$$

Where i and j is the indexing of the classes. When finding the derivative of Softmax there is two scenarios, which is when $j = i$ and when $j \neq i$.

The derivative is first found for the scenario: $j = i$.

$$sm_{j=i} = \frac{\partial \sigma(\boldsymbol{s})_i}{\partial \boldsymbol{s}_j}\bigg|_{j=i} = \left(\frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\right)' \tag{26}$$

Using the product rule and the chain rule yields the following.

$$\left(\frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\right)' = e^{\boldsymbol{s}_i}\frac{1}{\sum_j e^{\boldsymbol{s}_j}} - e^{\boldsymbol{s}_i}\frac{1}{\sum_j e^{2\boldsymbol{s}_j}}e^{\boldsymbol{s}_j} = \frac{e^{\boldsymbol{s}_i}e^{\boldsymbol{s}_j} - e^{\boldsymbol{s}_i}e^{\boldsymbol{s}_j}}{\sum_j e^{2\boldsymbol{s}_j}} = \frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\left(1 - \frac{e^{\boldsymbol{s}_j}}{\sum_j e^{\boldsymbol{s}_j}}\right) \tag{27}$$

For simplicity $\hat{\boldsymbol{y}}_i = \frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}$ and $\hat{\boldsymbol{y}}_j = \frac{e^{\boldsymbol{s}_j}}{\sum_j e^{\boldsymbol{s}_j}}$, the following can be written.

$$sm_{j=i} = \frac{\partial \sigma(\boldsymbol{s})_i}{\partial \boldsymbol{s}_j}\bigg|_{j=i} = \hat{\boldsymbol{y}}_i \left(1 - \hat{\boldsymbol{y}}_j\right) \tag{28}$$

The derivative is now found for the second scenario: $j \neq i$.

$$sm_{j\neq i} = \frac{\partial \sigma(\boldsymbol{s})_i}{\partial \boldsymbol{s}_j}\bigg|_{j\neq i} = \left(\frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\right)' \tag{29}$$

Using the chain rule yields the following.

$$\left(\frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\right)' = e^{\boldsymbol{s}_i}\frac{-1}{\sum_j e^{2\boldsymbol{s}_j}}e^{\boldsymbol{s}_j} = -\frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}\frac{e^{\boldsymbol{s}_j}}{\sum_j e^{\boldsymbol{s}_j}} \tag{30}$$

Once again for simplicity $\hat{\boldsymbol{y}}_i = \frac{e^{\boldsymbol{s}_i}}{\sum_j e^{\boldsymbol{s}_j}}$ and $\hat{\boldsymbol{y}}_j = \frac{e^{\boldsymbol{s}_j}}{\sum_j e^{\boldsymbol{s}_j}}$, allows the following to be written:

$$sm_{j\neq i} = \frac{\partial \sigma(\boldsymbol{s})_i}{\partial \boldsymbol{s}_j}\bigg|_{j\neq i} = -\hat{\boldsymbol{y}}_i\hat{\boldsymbol{y}}_j \tag{31}$$

Now that the two scenarios have been accounted for the derivative of the Softmax function can be written.

$$sm = \frac{\partial \sigma(\boldsymbol{s})_i}{\partial \boldsymbol{s}_j} = \begin{cases} \hat{\boldsymbol{y}}_i(1 - \hat{\boldsymbol{y}}_j), & \text{if } j = i \quad (32) \\ -\hat{\boldsymbol{y}}_i\hat{\boldsymbol{y}}_j, & \text{if } j \neq i \quad (33) \end{cases}$$

G.2 Deriving Combination of Softmax and Cross Entropy Loss

In section 4.2.1, equation 5, the cross entropy loss was simplified to:

$$L_i = -\boldsymbol{y}_i \log\left(\hat{\boldsymbol{y}}_i\right) \tag{34}$$

The derivative of the above is the following.

$$\frac{\partial L}{\partial \hat{\boldsymbol{y}}}_i = -\boldsymbol{y}_i \frac{1}{\hat{\boldsymbol{y}}_i} \tag{35}$$

Since softmax and the cross entropy loss is used in combination in the neural network, it makes sense to calculate a derivative of the loss with respect to the input of the Softmax function, which is $\boldsymbol{s}$. Remembering that $\hat{\boldsymbol{y}} = \sigma(\boldsymbol{s})$ and using the chain rule, the following

is written.

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}} \tag{36}$$

utilizing that softmax is composed of two cases, we get the following

$$\frac{\partial L}{\partial \boldsymbol{s}_i} = -\boldsymbol{y}_i \frac{1}{\hat{\boldsymbol{y}}_i} \hat{\boldsymbol{y}}_i (1 - \hat{\boldsymbol{y}}_i) + \sum_{j \neq i} \left(-\boldsymbol{y}_j \frac{1}{\hat{\boldsymbol{y}}_j} \right) (-\hat{\boldsymbol{y}}_i \hat{\boldsymbol{y}}_j) \tag{37}$$

The above can be simplified:

$$\frac{\partial L}{\partial \boldsymbol{s}_i} = -\boldsymbol{y}_i (1 - \hat{\boldsymbol{y}}_i) + \sum_{j \neq i} \boldsymbol{y}_j \hat{\boldsymbol{y}}_i = -\boldsymbol{y}_i + \boldsymbol{y}_i \hat{\boldsymbol{y}}_i + \sum_{j \neq i} \boldsymbol{y}_j \hat{\boldsymbol{y}}_i = \hat{\boldsymbol{y}}_i \left(\boldsymbol{y}_i + \sum_{j \neq i} \boldsymbol{y}_j \right) - \boldsymbol{y}_i \tag{38}$$

This can be written as.

$$\frac{\partial L}{\partial \boldsymbol{s}_i} = \hat{\boldsymbol{y}}_i - \boldsymbol{y}_i \Rightarrow \frac{\partial L}{\partial \boldsymbol{s}} = \hat{\boldsymbol{y}} - \boldsymbol{y} \tag{39}$$

This means that the gradient of the i'th score with respect to the loss is equal to the value after softmax, unless the i'th score is for the correct class, then it is the value after softmax subtracted with 1.