Plastic Bottle Cap Detection

Utilizing Artificial Intelligence

Kjetil Kveim

Thesis submitted for the degree of
Master in Electronics and Computer Technology - Cybernetics
30 credits

Department of Technology Systems
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019

Plastic Bottle Cap Detection

Utilizing Artificial Intelligence

Kjetil Kveim

Plastic Bottle Cap Detection

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

To interpret their surroundings, machines utilize sensors. Sensors provide information of the surrounding environment in the form of numeric values that are interpretable for computers. Computers can process the information received from the sensors. How the computers choose to process this information depends on the problem at hand. After processing the information, it is common for the computer to make a decision. Systems that are put together with sensors, a processing unit and other hardware, and with the purpose of solving specified tasks are called embedded systems.

Reverse vending machines (RVMs) are embedded systems that are designed to solve specific tasks. This masters thesis will explore the possibility of designing a solution that detects if the cap is attached to the plastic bottles that are inserted into the RVMs. The RVMs are equipped with six cameras. These cameras captures digital images as the plastic bottles are inserted into the RVMs. The digital images consists of numeric values which makes it possible for a computer to perform digital image processing.

The cap detection problem is a binary classification problem which consists of two classes; *Cap* and *No-cap*. There are three important criteria for this classification problem; Accuracy, classification time and size of classification algorithm. Accuracy is important to have a reliable classification. Classification time is important because the RVMs demands real-time classification. Size of the classification algorithm is important because RVMs have limited memory.
Pattern recognition, machine learning and deep learning are different approaches that are applied for the classification problem. In this study, the different approaches applied will be presented and compared.

The results of the different approaches applied vary. The best performance was achieved when deep learning was applied, more precisely convolutional neural networks (CNNs). CNNs finds relevant features in digital images on its own, unlike the other approaches where these features must be found manually. The features found in the CNNs proved to be good features and the networks managed to separate cap images from no-cap images with high accuracy. Some of the networks provided high accuracy as well as low classification time and small network size.

Sammendrag

Maskiner benytter seg av sensorer for å tolke omgivelsene sine. Sensorer gir informasjon i form av numeriske verdier som datamaskiner kan tolke. Hvordan disse sensordataene blir behandlet av datamaskinen varierer fra problem til problem. Når dataene er ferdig behandlet er det vanlig for datamaskinen å fatte en beslutning. Systemer som er satt sammen av sensorer, en prosesseringsenhet og annen maskinvare, som har som intensjon å løse en eller flere spesifikke oppgaver kalles innebygde systemer (embedded systems).

Pantemaskiner er innebygde systemer som er laget for å løse ulike spesifikke oppgaver. Denne masteroppgaven vil undersøke mulighetene for å lage en løsning som klarer å detektere om korken sitter på plastflaskene når de pantes. Pantemaskinene har seks kamerasensorer som tar bilder av flaskene når de puttes inn i åpningen på maskinen. Disse digitale bildene består av numeriske verdier som gjør det mulig for datamaskiner å prosessere dem.

Oppgaven kan ses på som et to-klasse problem med klassene: *Kork* og *Ikke-kork*. Det er tre viktige kriterier for klassifiseringsproblemet; Nøyaktighet, klassifiseringstid og størrelse på klassifiseringsalgoritmen. Nøyaktighet er viktig for å ha en pålitelig klassifisering. Klassifiseringstid er viktig siden pantemaskinen må kunne klassifisere i sanntid. Størrelse på klassifiseringsalgoritmen er viktig siden pantemaskinen har begrenset dataminne. Mønstergjenkjenning, maskinlæring og dyp læring er ulike tilnærminger som er anvendt for å løse klassifiseringsproblemet. I denne masteroppgaven vil disse tilnærmingene bli presentert og sammenlignet.

Resultatene for de ulike tilnærmingene varierer. Den beste løsningen ble oppnådd med dyp læring, nærmere sagt konvolusjonsnettverk. Disse konvolusjonsnettverkene finner egenskaper i bildene på egenhånd når de trenes. I de andre metodene må disse egenskapene trekkes ut av bildene manuelt. Egenskapene som ble funnet i konvolusjonsnettene viste seg å være bra egenskaper, og nettverkene klarte å separere de to klassene med høy nøyaktighet. Noen av nettverkene klarte å opprettholde høy nøyaktighet samtidig som klassifieringstiden var kort og nettverksstørrelsene små.

Preface

This master thesis is written as a part of the Masters degree in cybernetics at the university of Oslo, UiO. The task was handed out by Tomra ASA.

I would like to thank my family for great support troughout my education. I would like to thank my supervisor Idar Dyrdal from UiO for always taking the time to help and for good guidance. I would also like to thank Johnny Njåstad, Willy Olafsen and Morten Løken at Tomra for all help during this thesis and for the provided office space.

Contents

List of Figures

List of Tables

Abbreviations

RVM - Reverse vending machine

AI - Artificial intelligence

CNN - Convolutional neural networks

ROI - Region of interest

CPU - Central processing unit

GPU - Graphics processing unit

SVM - Support vector machine

SGD - Stochastic gradient descent

SGDM - Stochastic gradient descent with momentum

ILSVRC - ImageNet Large-Scale Visual Recognition Challenge

1 Introduction

This section will provide an introduction for this thesis. The motivation for solving the task at hand will be presented, as well as the working material and tools used to solve the task.

1.1 Motivation

Plastic waste pollution is an increasing problem worldwide. Over 1 million plastic bottles are purchased every minute and less than half of these are recycled [15]. TOMRA was founded in 1972 and creates solutions for recycling. The reverse vending machine has been developed by TOMRA since 1972 and is an important product for decreasing both plastic and metal waste. As of today the RVMs do not detect whether or not the plastic cap is attached to the plastic bottle when recycled. It is desirable to also recycle the plastic cap of every plastic bottle to maximize the amount of plastic that are recycled.
This master thesis will utilize artificial intelligence to create different image classifiers that detect if the cap is attached to the plastic bottle. Different approaches such as pattern recognition, machine learning and deep learning will be applied and the results of these methods will be compared. The problem can be thought of as a two class problem, where the corresponding two classes are; *Cap* and *No-cap*. The two-class classification problem is also called binary classification. In binary classification a classifier predicts one out of two classes on the basis of a decision rule. Images of plastic bottles on the belt in the reverse vending machine are the basis for which these approaches will be applied.

1.2 Work materials and tools

To solve the task, images are used to gather information from the plastic bottles. 6 different cameras capture images of the plastic bottles as they enter the RVM. These images are used to create different classifiers. MATLAB is used throughout this thesis for creating the different classifiers. The different toolboxes provided by MATLAB that are applied during this thesis are *Computer Vision System Toolbox*, *Deep Learning Toolbox* and *Image Processing Toolbox*.

2 Theory

2.1 Artificial intelligence approaches and their differences

In figure 1 it is shown that artificial intelligence is a broad term. AI can be thought of as a machine with some intelligence that imitates decisions of humans. This can either be done using simple statements and rules, or in a more extensive way which allows for the machine to learn based on a set of input features. AI covers, but is not limited to, classification problems.

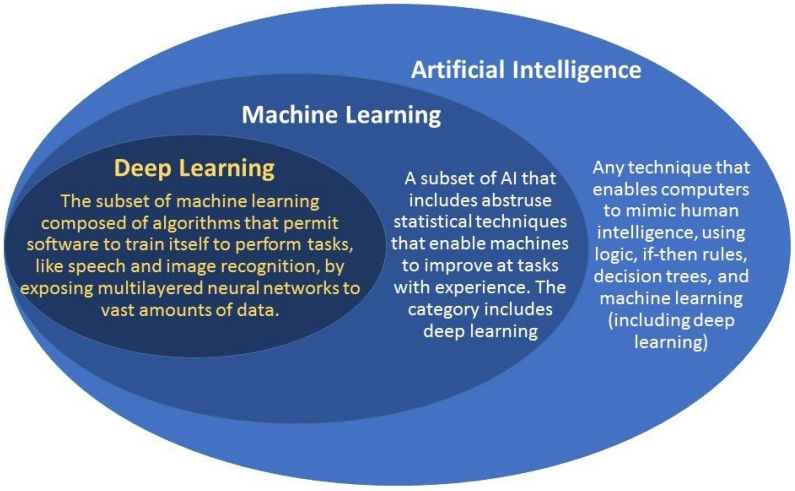


Figure 1: Illustration of the term artificial intelligence. Image courtesy of [9].

2.1.1 Pattern recognition

Pattern recognition is a sub area of artificial intelligence and there are several approaches towards pattern recognition. Machine learning is one way where a machine learn a decision rule based on input features. Another way is to manually assign decision rules to the machine. Methods within computer vision are often used. For instance if there is a certain pattern in an image, matching algorithms like template matching can be applied, or algorithms which tries to find e.g edges or elliptic patterns. For some cases, simple logic like an if-statement is sufficient to differentiate between classes.

2.1.2 Machine learning

A machines ability to learn relies on access to a dataset. This dataset consists of many samples with corresponding features. A feature is a numerical value which describes your sample in some way. The feature space can be of different dimensions. For dimensions larger than 3D it is difficult for the human eye to visualize the samples, and the strength of machines really shines through. The idea is that a machine learns what this data represents and exploits its knowledge to predict or classify a new sample.

2.1.3 Deep learning

Deep learning is a branch within machine learning. Inspiration is taken from the actual human brain and how neurons in the brain interacts. Deep neural networks are vaster than more traditional methods within machine learning and often demands a much larger dataset and are more computationally expensive.
When the dataset consists of images, a natural way would be to apply a convolutional neural network. The input feature vector will consist of the raw image pixels and not manually selected features. This will result in a very high dimensional feature space. For instance, a 40x40 pixel image would consist of 1600 input features.
In other words, a convolutional neural network learns directly from images and the need of manually extracting input features are eliminated [4]. When a convolutional neural network is trained, a set of weight vectors are stored and will be used to classify a new image.

2.2 Digital images

Digital images are represented as pixels in a spatial domain with a certain number of channels. This spatial domain can be thought of as rows and columns in a matrix or as x-axis and y-axis in a coordinate system. There is one pixel at each element in the spatial domain. These pixels are represented by a numeric value that describes the intensity of each channel. For grayscale images there is only one channel, the gray level, and the digital image is a two dimensional matrix. For RGB images there are 3 channels, where each channel represent the presence of either the color red, green or blue. These RGB images are three 2D matrices which can be stacked and represented as one 3D matrix. In figure 2, one gray scale and one RGB image is visualized. A pixel in the grayscale image is represented as shown in equation 1 and a pixel in the RGB image is represented as shown in equation 2.

$$[row,\ column] = [gray\ intensity] \tag{1}$$

$$[row,\ column] = [red\ intensity,\ green\ intensity,\ blue\ intensity] \tag{2}$$

The pixel intensity values normally ranges from 0 to 255, where 0 is no presence of intensity and 255 is max presence of intensity. The pixel intensity range may vary, e.g the pixel intensity values in binary images are represented by just 1 bit and ranges from 0 to 1. If all channels in the image have the minimum pixel intensity value, the pixel is black. If all channels in the image have maximum pixel intensity value, the pixel is white. Colors (for RGB images) and graylevel (for grayscale images) are manipulated by adjusting the pixel intensity value in the channels.

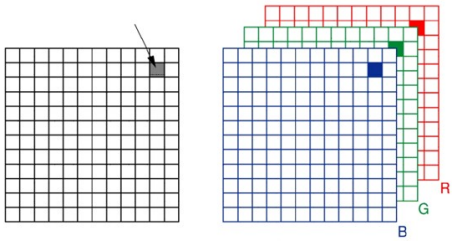


Figure 2: Visualization of grayscale image (left) and RGB image (right). The array points towards one pixel. Image courtesy of [21].

2.2.1 Image histogram

The pixels in a digital image can be represented by a histogram. For grayscale images, which only have one channel, the pixel intensity values are represented along the x-axis and the amount of each pixel intensity value along the y-axis. In figure 4, the image histogram for the image in figure 3 is shown. In the image histogram plot, the peaks around pixel intensity value 30 represents the black background and the other black pixels in the image. The peak at pixel intensity value around 80 represents the gray plastic bottle cap.

Figure 3: An image of a plastic bottle with cap

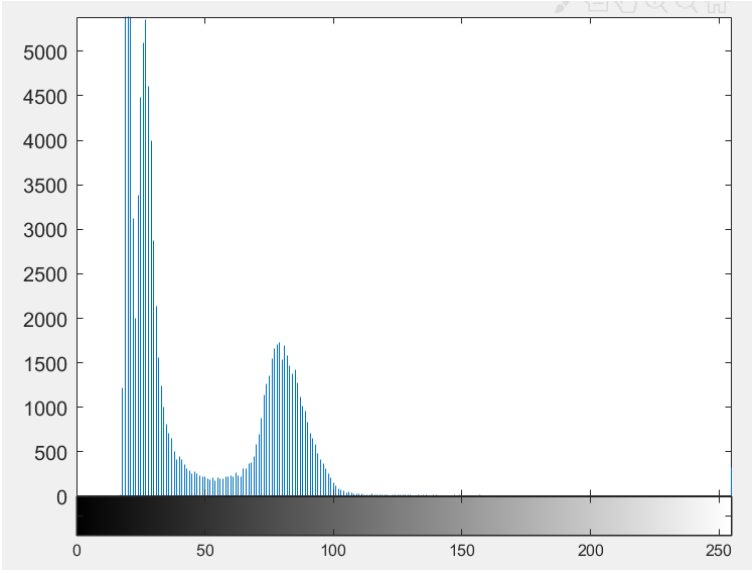


Figure 4: The corresponding image histogram for the image in figure 3

2.3 Digital image processing

A digital image can be considered as a two dimensional digital signal. These signals can be processed and this is further referred to as image processing. When processing an image, useful information from the image can be extracted or the image can be altered.
Image processing can be used to resize images. This is done by altering the number of pixels from the original image. Making the number of pixels lower, results in a smaller resolution image and are called downsampling. An image can also be interpolated to increase the number of pixels. This is called upsampling. There are several different ways to perform image resizing. By default, the imresize function provided by MATLAB uses bicubic interpolation with antialiasing.
Other common operations in image processing are to extract corners and edges, and to blur or sharpen an image. These operations can be done by applying a filter mask or kernel and convolve this kernel with the image.

2.4 Kernels

A kernel is a filter mask that can be used to convolve with the input image. The kernel is often a small square matrix. The number of rows and columns are often odd, so that the kernel have one element in the center. The output image can be an enhanced version of the input image in some way or an image which contains specific features of the input image. The element values in the kernel matrix decides the effect of the convolution. Some common applications are image smoothing, noise reduction, image sharpening and edge detection.

2.5 Convolution

As stated above, an image can be thought of as a two dimensional digital signal. A two dimensional convolution can be performed on the spatial domain of an image. When a kernel is convolved with the input image, the pixels in the output image are given by a linear combination of the pixels in a local neighborhood in the input image [12].

The mathematical definition of the convolution can be seen in equation 3 [12]. The output image is O, the input image is I with spatial dimension $M\ x\ N$ and the kernel is K with spatial dimension $2m+1\ x\ 2n+1$. Equation 3 shows how one pixel in the output image is calculated. Equation 4 shows how a convolutional operation is written symbolically.

$$O(i,\ j) = \sum_{u=-m}^{m} \sum_{v=-n}^{n} I(i-u,\ j-v)K(u,\ v) \tag{3}$$

$$O(i,\ j) = (K \star I)(i,\ j) \tag{4}$$

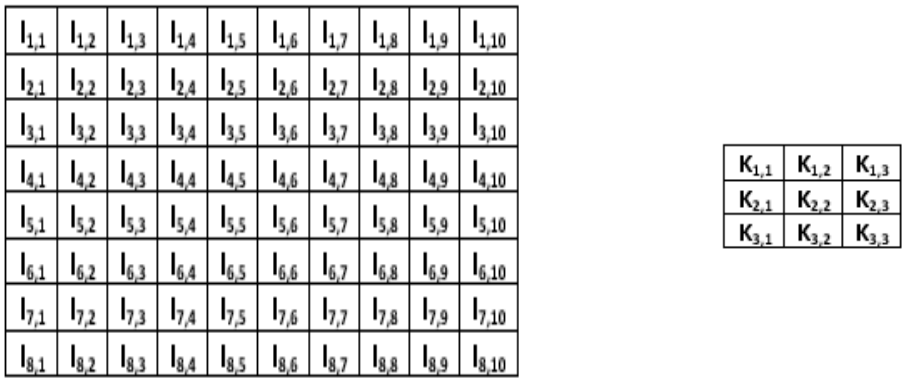


Figure 5: Visualization of input image (left) and kernel (right).

The spatial dimension of the output image O is determined by the number of iterations equation 3 is executed. If it is important to maintain the same spatial dimension as the input image, pixels must be added to the image along the borders. This is called padding. These pixels must be added such that the kernel fits around all pixels in the input image. Without padding the kernel does not fit the pixels in the input image along the borders. The size of this padding is decided by the spatial dimension of the kernel. The pixels that are added with padding are often 0, and this is called zero-padding. In figure 6 the input image from figure 5 is shown with zero-padding. It is also shown that a kernel with spatial dimension of 3 x 3 now fits the pixels in the input image along the border. In the figure the kernel is placed around pixel I(8, 1) in the input image. The kernel center now fits all the true pixels of the input image and the spatial dimension is preserved in the output image. Zero padding is a widely used method, but it is not the only padding option. The padding can also consists of its neighbouring pixel, i.e the pixels at the border of the image are copied in the padding.

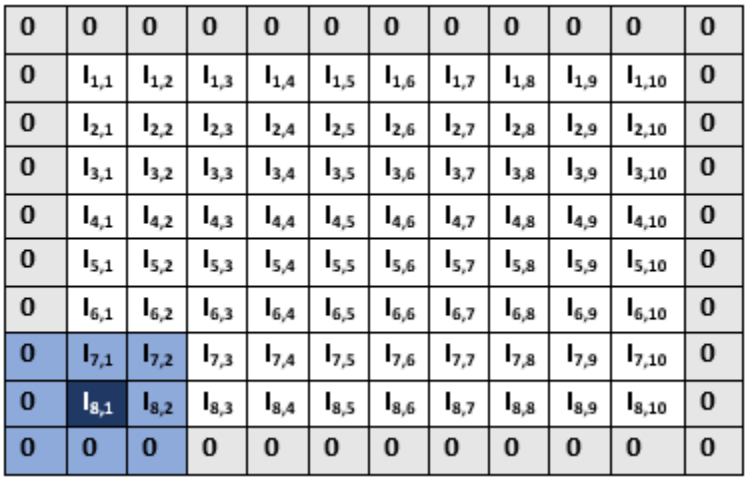


Figure 6: Input image with zero padding, to preserve spatial dimension. The kernel center is placed on top of every image pixel.

However, the pixels added with padding are just made up to preserve the spatial dimension of the input image and would result in some distortion in these regions. To prevent these distortion regions it is possible to just execute equation 3 with a clipped input image. This will not keep the spatial dimension, but will get rid of distortion regions due to padding. When the kernel convolves with the input image, all the pixels that are in the local neighborhood are true pixel values. In figure 7, it is shown how the *3 x 3* kernel only centers around pixels that are not on the border of the input image. Equation 3 is executed from $i = 1$ to $M - m + 1$ and $j = 1$ to $N - n + 1$, where the spatial dimension of the input image is M x N and the spatial dimension of the kernel is m x n. The output image will get the spatial dimension of $(M - m + 1)$ x $(N - n + 1)$.

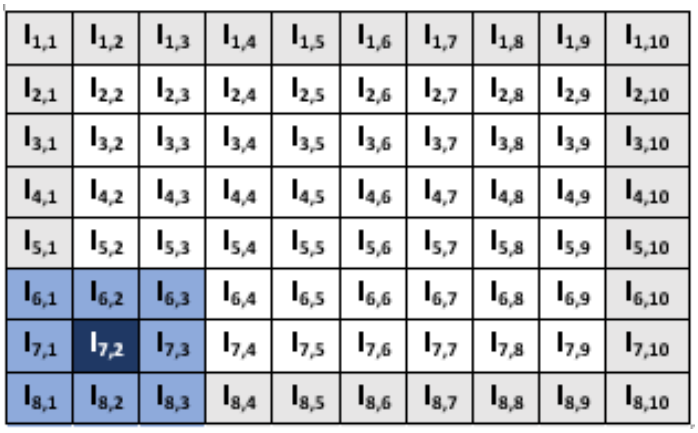


Figure 7: The kernel center is not placed on top of the border pixels.

2.6 Canny edge detection

The canny edge detection method [23] is an algorithm which can be divided into 4 steps. The 4 steps are shown in the list below and each step will be gone through separately.

- Noise reduction
- Gradient of image intensity function
- Non-maximum suppression
- Thresholding - connectivity in edges

2.6.1 Noise reduction

Noise in the image can be troublesome for the edge detection algorithm. Therefore the first stage of the Canny edge detection algorithm is to remove unwanted noise from the image. Noise is removed by Gaussian filtering [31] which smooths the image and removes unwanted details.
The Gaussian function in one dimension with zero mean, is given by equation 5. σ is the standard deviation of the distribution.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{5}$$

The one dimensional Gaussian distribution with $\sigma = 2$ is visualized in figure 8. There are just 2.2 % of the Gaussian distribution that lies outside the range of ± 3 standard deviations. One standard deviation is denoted as σ.

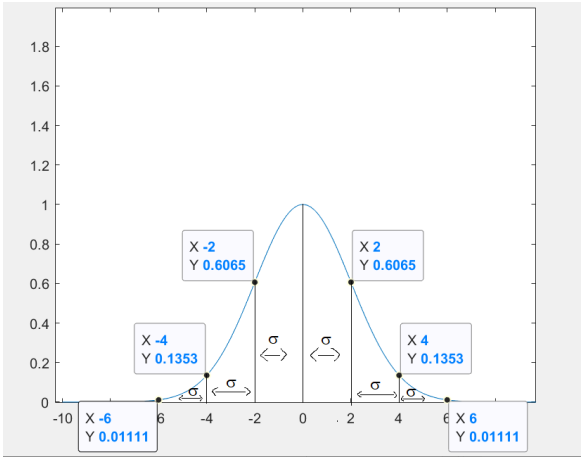


Figure 8: One dimensional Gaussian distribution with $\sigma = 2$ and $\mu = 0$.

Since digital images can be considered as two dimensional digital signals, we need a two dimensional Gaussian distribution. This can be obtained by multiplying a one dimensional Gaussian distribution in x-direction with a one dimensional Gaussian distribution in y-direction. The two dimensional Gaussian distribution with a diagonal covariance matrix [11] is shown in equation 6 and visualized in figure 9 with $\sum = \begin{matrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{matrix}$ and $\mu = \begin{matrix} 0 \\ 0 \end{matrix}$. For simplicity σ_1 and σ_2 are set to be equal in equation 6.

$$G(x,\ y) = G(x) * G(y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{6}$$

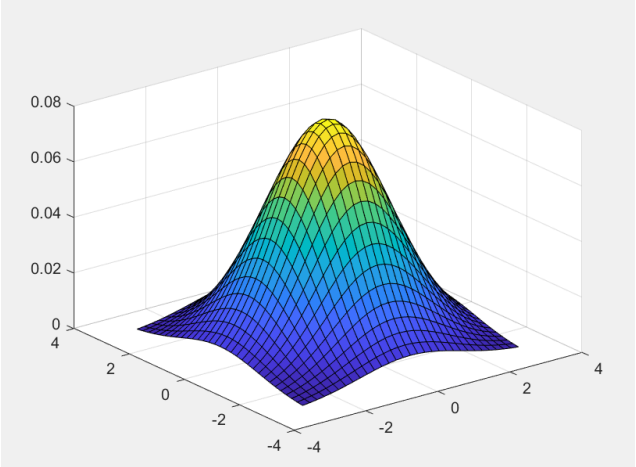


Figure 9: Two dimensional Gaussian distribution.

The two dimensional Gaussian distribution needs to be discrete as digital images consist of pixels. The discrete result will then work as an approximation of the Gaussian function. It was stated earlier in this section that in a distance of ±3 standard deviations from the center only 2.2 % of the distribution is missing. This can, for practical purposes, be considered as 0. The canny edge detection uses a 5 x 5 Gaussian kernel. This kernel is the discrete approximation of the Gaussian distribution. For a Gaussian function with a standard deviation of 1 for both x and y, this kernel will be represented as shown in figure 10 [31]. This kernel will be used to convolve with an image and the output image will be a blurred or smoothed version of the input image. This will result in noise reduction.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 10: 5 x 5 Gaussian kernel, $\sigma = 1$.

2.6.2 Gradient of image intensity function

The Sobel operator is a commonly used operator for detecting edges and is deployed in the Canny edge detection algorithm. Edge detection is performed after the process of reducing noise in the image. Noise may appear as sudden changes in the intensity of pixels and will cause unwanted edges. The Sobel operator in itself also perform noise suppression.
Two 3 x 3 kernels, one for horizontal edges and one for vertical edges, forms the Sobel operator. These two kernels are shown in figure 11 and are discrete approximations of the Gaussian function in horizontal and vertical direction. The leftmost kernel finds vertical edges and the rightmost kernel finds horizontal edges.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 11: The two kernels included in the Sobel operator. Vertical edges kernel (left) horizontal edges kernel (right).

Edges are located in regions where the intensity in image pixels have large changes. After smoothing with the Gaussian filter has been applied, the Sobel operator computes an approximation of the first derivative of an image intensity function [27]. In equation 7 through equation 9 the two kernels included in the Sobel operator are referred to as K_x and K_y. Where K_x are the left kernel in figure 11 and K_y the right. G_x is the approximated gradient in x-direction, G_y is the approximated

gradient in y-direction and G is the magnitude of G_x and G_y combined. I is the input image.

$$G_x = K_x \star I \tag{7}$$

$$G_y = K_y \star I \tag{8}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{9}$$

Equation 10 shows how the angle of the direction the gradient points in each pixel is computed. θ is the angle orthogonal to the edge.

$$\theta = tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{10}$$

After convolving the two Sobel kernels over the smoothed image from the previous step, edges are now represented with a strength and a direction. The strength of the edge is represented as the magnitude G. And the direction of the gradient is represented as the angle θ. These two values forms the gradient at each pixel in the image.

2.6.3 Non-maximum suppression

Non-maximum suppression is used to thin out the edges produced by the image gradients. This is done by keeping the maximum gradient magnitude across the edge. The maximum edges in a neighborhood are kept and all other edge pixels in the same neighborhood are set to zero. Non-maximum edges are suppressed resulting in an image where just the largest edges are kept.

2.6.4 Thresholding - connectivity in edges

Two numeric threshold values are introduced. A minimum threshold value and a maximum threshold value. All edges with a magnitude G that are below the minimum value are considered not edges. Likewise all edges that are above the maximum value are considered edges. The interesting part is what happens to edges that have a gradient magnitude value between the two thresholds. An edge with gradient magnitude value between these thresholds are considered an edge if, and only if, this edge is connected to an edge with gradient magnitude level above the maximum threshold value. In other words; Very weak edges are discarded, very strong edges are kept, and edges in the middle are kept only if they are connected to a very strong edge, otherwise they are discarded.

2.7 Machine learning

Machine learning is a sub area within artificial intelligence. The idea behind machine learning is to create an application which learns from experience. To allow this application to learn anything at all, a dataset must be provided. This eliminates the need of explicitly creating algorithms to predict the outcome of new data samples. This is taken care of by the machine learning application.

2.7.1 Supervised and unsupervised learning

Machine learning can be applied in either a supervised approach or in an unsupervised approach. Supervised learning is applied when having labeled data, i.e a dataset with known output values. Unsupervised learning is performed when the dataset only consists of input values with no known labels.

Supervised learning: Supervised learning is commonly used for classification or regression problems. When the labels are represented as categories, such as *cap* and *no-cap* the problem is a classification problem. When the labels are represented as a numeric value, such as a prices in NOK, heights in centimeters and so on, the problem is a regression problem.

unsupervised learning: In the unsupervised learning approach the dataset only consists of input features. The labels are absent and the machine learning algorithms can not exploit the fact of knowing the true labels during the learning process. Unsupervised learning approaches look for structure in the dataset.

Semi-supervised learning: Semi-supervised learning is a case where both supervised and unsupervised learning are applied. Some of the data samples in the dataset are labeled, but not all. This is a common approach, especially when there is limited access to labeled data. Labeling data samples are often very time consuming, therefore it may be more suitable to just label some of the data.

For this thesis, supervised learning is the only machine learning approach that is applied, since the dataset is labeled. This is the approach that will be considered further in this thesis when machine learning and deep learning are involved.

The dataset D consists of input features X and corresponding output values Y as seen in equation 11.

$$D = (X, Y) \tag{11}$$

Further, a machine learning algorithm is applied. This algorithm learns a mapping function which yields a set of weights. This weight vector is a mapping function which maps the input to an output. The mapping function can be seen in equation 12. $\hat{Y}$ is the predicted output value, X is the input data and f is the mapping function.

$$\hat{Y} = f(X) \tag{12}$$

The mapping function is learned through experience. The machine learning algorithm iteratively learns by applying the weight vector to the input features, present a prediction and adjust the weights accordingly. This can be done since the dataset consists of true labels which can be exploited to tell how well the machine learning algorithm performs during the learning process.

2.7.2 Traditional machine learning

During this thesis, traditional machine learning is the term used when referring to simpler machine learning approaches where the features needs to be manually extracted. This means that the dataset can not consist of raw image pixels. Numerical features extracted from the image forms the dataset used in this approach. The training time and complexity of the machine learning algorithms are smaller and simpler, but the resources spent in extracting features can be much larger. Support vector machine (SVM) is a well known approach. For binary classification, the idea behind SVM is to find a hyperplane that manages to fully differentiate between the two classes. This hyperplane is the hyperplane that have the largest distance between the two classes. The optimal hyperplane is located at the maximum orthogonal distance to the closest data sample from each class is [29]. This is the margin visualized in figure 12. In this figure the concept of SVM is visualized. The data samples closest to the hyperplane are called support vectors.
Separable data is not always the case. For non-separable data SVM uses a soft margin. This results in a hyperplane that does not separate every data sample.

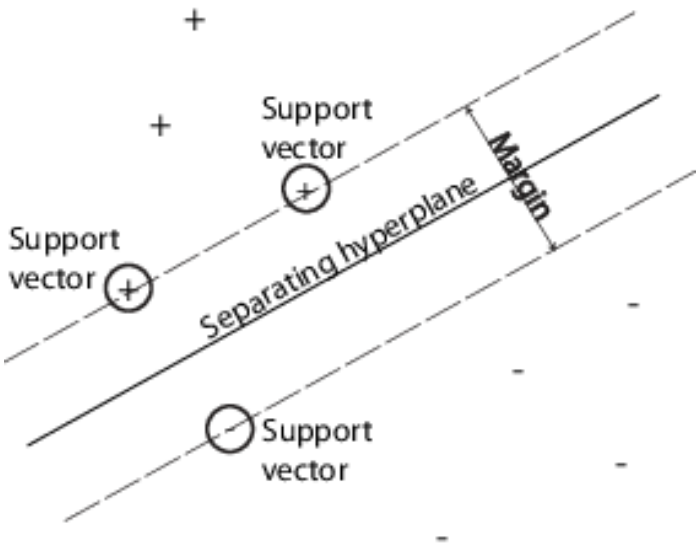


Figure 12: The concept of SVM. + and - indicates data samples from the two classes. Image courtesy of [29].

2.7.3 Machine learning and deep learning

Although deep learning is a sub area within machine learning there is a big difference between deep learning algorithms and traditional machine learning algorithms. Unlike traditional machine learning, deep learning can handle a dataset consisting of raw image pixels. Figure 13 describes the main difference between traditional machine learning and deep learning. Feature extraction is done manually for traditional machine learning and automatically for deep learning.

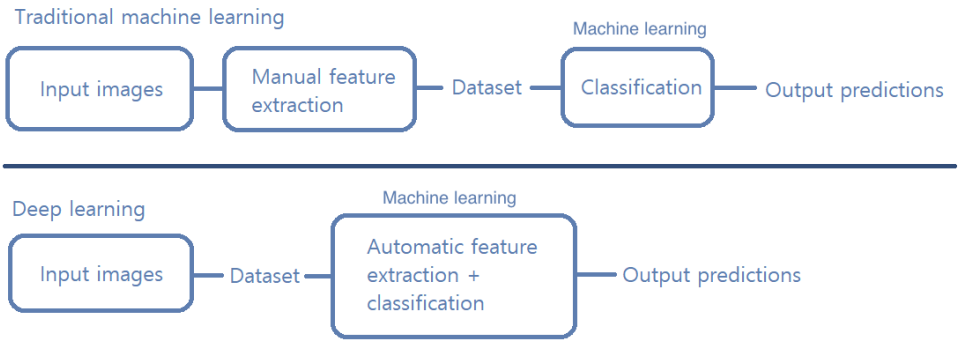


Figure 13: The difference between traditional machine learning and deep learning.

2.8 Deep learning

Deep learning is a subcategory of machine learning. Both deep learning and machine learning learns by examples, but deep learning normally requires a lot more data and the networks are much bigger. Deep learning was introduced in the 1980s by Rina Dechter [8], but have only in more recent years been popular and applicable to many problems. In the 1980s the computational resources that are present today were not available. Deep learning methods are putting into use the computing power that are available as of today. GPUs allows for parallel computing and are especially profitable for deep learning. In others words, today's hardware makes it possible for deep learning methods to learn fast and perform with high accuracy.
For this thesis the focus is on the neural network architectures of deep learning. Deep neural networks tries to imitate how the human brain receives inputs and make decisions. The deep learning approaches models the concept of how the brain works in very simplified versions, but yet the deep learning approaches outperforms humans in some problems.

A small neural network is visualized in figure 14. The circles in the figure are called nodes. One layer consists of several nodes. In the figure there are shown three layers, the red input layer, the blue hidden layer and the green output layer. All of the nodes are connected to each node in the next layer, this is what is called a fully connected network. The term *deep* from deep learning comes from the number and size of the hidden layers. In most networks these hidden layers are the main part of the network. The input layer are the features from the dataset and the output layer calculates scores of the different classes.
For image classification problems these fully connected networks would result in too many parameters. Imagining an image of size 250x250 and many hidden layers where all nodes are connected to nodes in the next layer, the network would be gigantic. Therefore convolutional neural networks are a better approach for image classification problems. The convolutional layers do not connect each node to every node in the next layer. Only a selection of nodes are connected to the next layer which results in fewer parameters.

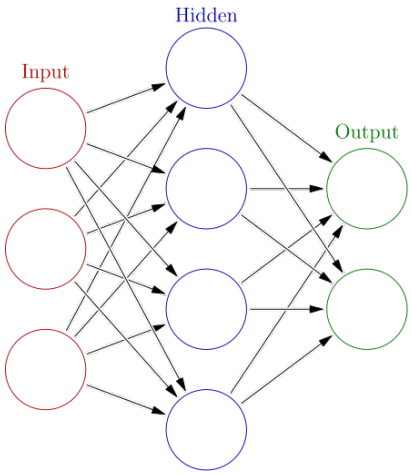


Figure 14: Nodes and layers of a neural network. Image courtesy of [16].

2.9 Convolutional neural networks

Convolutional neural networks, abbreviated to CNNs, are one category of deep neural networks. CNNs have proven to be good when dealing with image classification problems. One of the most important perks of CNNs is that it requires very little preprocessing. The filters which must be manually made to extract useful information from the images in other approaches, will in CNNs be found in what is called the hidden layers of the network. This means that the whole process of extracting features are eliminated, as stated earlier. What happens in the different layers of the network and how the network learns will be briefly explained in the following subsections.

2.9.1 Architecture

The model of the CNN describes what layers that are used, how they are combined and what hyperparameters each layer has. This model is called the architecture of the CNN. Even though there are many ways of constructing a model, there are some layers that are especially common. Convolutional layers, pooling layers and fully connected layers are examples of some of the most common layers. These layers and more will be explained further in the subsections to come. These layers are included in what is called the hidden layers. In figure 15 a CNN model is shown. The

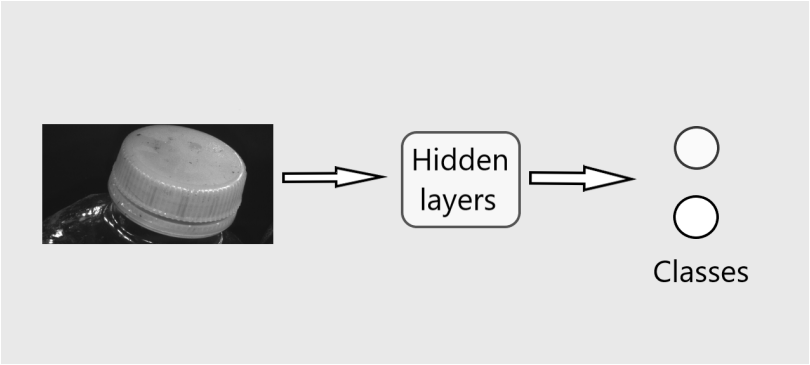


Figure 15: CNN model with input image, hidden layers and class probabilities.

input image is directly fed to the hidden layers of the model which generates class affiliations in terms of class scores. These layers are denoted as hidden due to the lack of human interaction during training. These act as a feature extractor for the network and decides which features that are important for the different classes in the dataset. The inputs to the network are image pixels and require very little consideration in terms of preprocessing. The outputs are class scores and the class with the highest score is the predicted class.
The interesting part, and what decides the characteristics of the CNN lays within the hidden layers. Some of the most common layers found inside these will be explained further in section 2.9.2.

2.9.2 Layers

The hidden layers of a network consists of several different layers. Some of these layers have learnable parameters which updates its parameters at each iteration and some of these layers facilitates the model in other ways. Some layers have hyperparameters, which are parameters that can be set before learning begins, i.e hyperparameters are not learnable and need to be manually assigned. Common types of layers will be described below.

Convolutional layers: Convolutional layers applies kernels and performs convolutional operations. The convolutional operation is described earlier in section 2.5. The convolutional layers performs convolutional operations on the previous layer in the model. There are several hyperparameters that can be set for this layer. The filter size, stride, zero-padding and dilation are all hyperparameters that needs to be set and will determine the output size and the characteristics of the layer. The key hallmark for a convolutional layer is that only a selection of nodes from the previous layer are connected to a node in the next layer.
The *filter size* is the size of the kernel that convolves with the input. There can be applied multiple filters in one convolutional layer. The number of filters assigned will increase the depth dimension of the model.
Stride is another hyperparameter that decides how many pixels the kernel moves each iteration. If stride is set to 1 the kernel moves one pixel at a time, if stride is set to 2 the kernel moves two pixels at a time and so on. The stride decides how many pixels the kernel moves both horizontally and vertically.
Zero-padding helps to control the spatial dimension of the output. This was explained earlier in section 2.5. For the case of CNNs, zero-padding is commonly used to preserve the width and height of the input in the output.
Filter size, stride and zero-padding are the hyperparameters that determines what the dimension of the output will be. The outputs spatial dimension of a convolutional layer can be calculated according to equation 13. O is the spatial dimension of the output, I is the dimension of the input, F is the width and height of the filter, P is the zero-padding size and S is the stride. Equation 13 assumes that the spatial dimension of the input is square, i.e width and height are the same. If the inputs spatial dimension differs in width and height, equation 13 must be calculated twice, both for the height dimension and for the width dimension.

$$O = \frac{I - F + 2P}{S} + 1 \tag{13}$$

Batch normalization layers: Batch normalization layers are often used after convolutional layers. These layers normalizes each mini-batch of data that flows through the model during training. The batch normalization layers speeds up training and help prevent the case of overfitting. Overfitting is explained in 2.8.6. The layer normalizes the output of the previous layer by subtracting the mean of the mini-batch and divide by the standard deviation of the mini-batch. Batch normalization can be thought of as normalizing the data after each hidden layer. In equation 14 the normalization of a batch is shown. μ_i and σ_i^2 are the mean and variance of the mini-batch and x_i is one mini-batch of data. ϵ is a small numeric constant to help numerical stability in case the variance is very close to zero, which will in practice result in division by zero [2]. The normalized output of the mini-batch is $\hat{x_i}$ and have a unit Gaussian distribution.

$$\hat{x_i} = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{14}$$

μ_i and σ_i can be found by equation 15 and 16. In these equations, m is the size of the mini-batch and ϕ is a small positive numeric constant [1].

$$\mu_i = \frac{1}{m} \sum_{j=1}^{m} x_{ij} \tag{15}$$

$$\sigma_i = \sqrt{\phi + \frac{1}{m} \sum_{j=1}^{m} (x_i - \mu_i)_j^2} \tag{16}$$

After calculating equation 14 the batch normalization layer shifts and scales the normalized output $\hat{x_i}$ by two learnable parameters γ and β. These parameters are randomly initialized and updated during training. Mini-batches contain small amount of data and can vary a lot. Batch normalization is included in the training process and the weights will get unbalanced updates if the mini-batches vary a lot. By shifting and scaling the normalized output $\hat{x_i}$ by the learnable parameters, a new optimized mean and standard deviation is set. This process is shown in equation 17. y_i is the new scaled and shifted normalized output. $\hat{x_i}$ is the normalized output calculated in equation 14 [2].

$$y_i = \gamma \hat{x_i} + \beta \tag{17}$$

When training is complete the trainable mean and variance are set respectively to the mean of the entire training set and the variance of the entire training set [2].

Activation functions: Activation functions are applied after convolutional layers and fully connected layers. If batch normalization layers are present, the activation function is normally applied after these as well. The purpose of the activation function is to add non-linearity to the model, therefore activation functions are also referred to as non-linearities. In this thesis, activation functions will be the used term. The benefit of using an activation function and making the model nonlinear is that this makes it possible to stack several layers. If a model is linear, all layers can be thought of as a single layer due to the fact that a combination of linear functions is just a new linear function. Several different activation functions exists, but the most used activation function as of today is the ReLU activation function. ReLU (rectified linear unit) is given by equation 18 and visualized in figure 16.

$$f(x) = max(0, x) = \begin{cases} 0, & \text{for x} < 0 \\ \text{x}, & \text{for x} \geq 0 \end{cases} \tag{18}$$

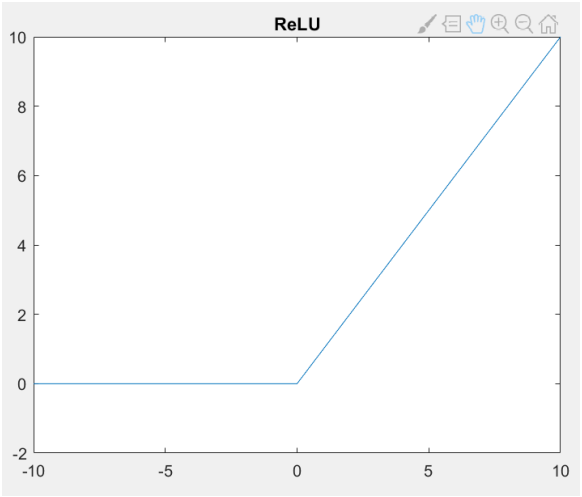


Figure 16: Plot of the ReLU activation function.

Pooling layers: Pooling layers are used for subsampling. It is common to apply a pooling layer after a convolutional layer. Subsampling is performed to downsample the input in the spatial dimension. There are multiple ways of reducing the spatial dimension of the input. Max-pooling and average-pooling are two examples. Filter size and stride are chosen for the pooling layer. The stride is often set to be equal to the filter size such that the filter does not overlap previously used values when it iterates over the input. Max pooling chooses the maximum value inside the filter at each iteration. Average pooling chooses the average of the values inside the filter at each iteration. In figure 17, max pooling is described. Here, the filter size is 2 x 2 and the stride is set to 2, so that the filter skips values that are already considered at the previous locations of the filter. A pooling layer with a 2 x 2 filter and stride equal to 2 will produce an output which has halve the size of the input in both spatial directions. As shown in figure 17 a 4 x 4 input is downsampled to a 2 x 2 output, resulting in reducing the spatial size of the input by a factor of four.

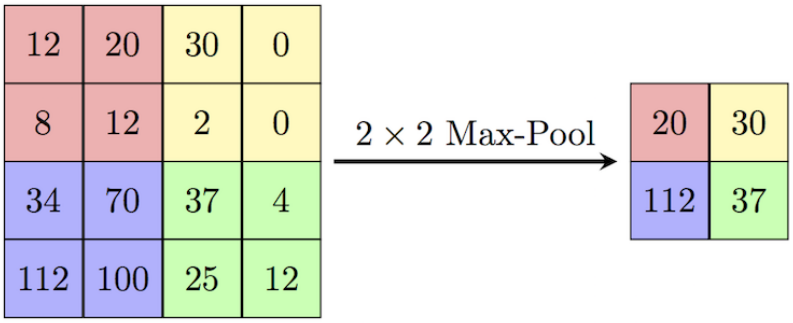


Figure 17: Max pooling with filter size 2 x 2 and stride equal to 2. Image courtesy of [22].

Equation 19 describes how the spatial dimension of the output is calculated for a pooling layer. O is the spatial dimension of the output, I is the spatial dimension of the input, P_s is the size of the pool filter and S is the stride. It is worth mentioning that the depth dimension is preserved during a pooling layer.

$$O = \frac{I - P_s}{S} + 1 \tag{19}$$

Fully connected layers: Fully connected layers are also called dense layers, and are often to be found at the end of CNNs. In fully connected layers all the input nodes are connected to each node in the layer. The main difference between a fully connected layer and a convolutional layer is that the neurons in a convolutional layer are only connected to a local region of the input, while in a fully connected layer all neurons are connected to each of the input neurons [5]. The input needs to be flattened before the fully connected layer, this means that it needs to be represented as a vector. In a classification problem the last fully connected layer of the model needs to be of size 1 x 1 x C, where C is the number of classes.

2.9.3 Receptive field

The receptive field is also known as the field of view, but in this thesis the term receptive field will be used. The receptive field can briefly be explained as how much one node in a layer sees. The receptive field is the information from nodes in previous layers that are available for a node in the next layer. All areas outside the receptive field will not be taken into consideration for the current

node. It is important to consider the receptive field when creating a CNN and to keep the size of the receptive field at balance, such that important information is kept.
The impact of the receptive field on one specific node can in many cases be proven to have a Gaussian distribution [20]. In other words, each node has a receptive field consisting of nodes from the previous layers and all these nodes are connected, but the nodes located at the center of the receptive field are connected to more nodes included in the receptive field than the nodes located furthest out in the receptive field. The connections of the nodes in the receptive field will form a Gaussian distribution. Gaussian distributions often decay quickly from its center and the effective receptive field is introduced as the area of this Gaussian distribution which can be considered. The effective receptive field only represents a fraction of the theoretical receptive field and is a better measure of the region that impacts a node in the next layer.
There are several ways to increase the receptive field of the network. Adding convolutional layers, adding stride to the convolutional layers, adding a dilation factor to the convolutional layers and adding pooling layers are common methods for increasing the receptive field.
Adding convolutional layers to the model will both increase the receptive field and the number of parameters in the network. For a convolutional layer with filter size $k \ x \ k$, the receptive field will grow by $k\text{-}1 \ x \ k\text{-}1$ for each layer. Figure 18 illustrates how the receptive field increases by a factor of two for each convolutional layer when applying filters with size $3 \ x \ 3$. The receptive field are the number of input nodes that are highlighted in purple. In figure 18 the receptive field for a node in the third hidden layer is 7.

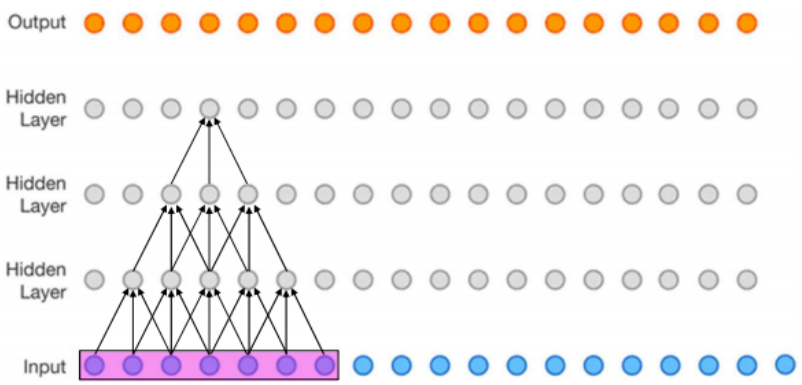


Figure 18: Receptive field for convolutional layers with filter size 3 x 3. Image courtesy of [18].

By adding stride to the convolutional layers the receptive field will increase more quickly. The concept of stride can be found in section 2.9.2. By adding stride, positions in the input layer are skipped which makes the model able to cover a larger area. In figure 19, stride of 2 is added to the first hidden layer of the network. Which results in a receptive field of 7 already at hidden layer 2. In figure 18, where the stride was one, a receptive field of 7 was achieved one layer later, in hidden layer 3.

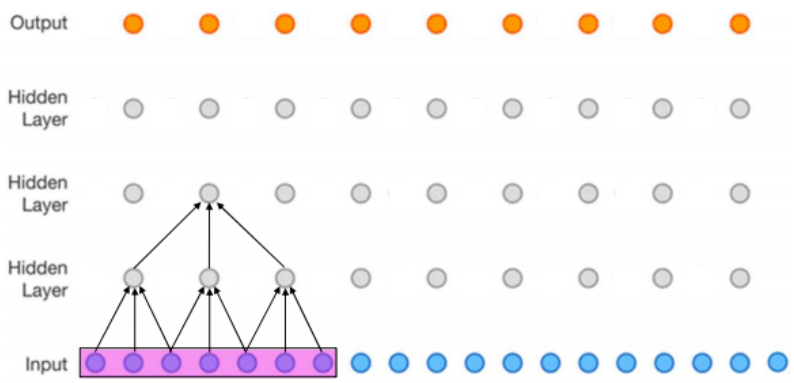


Figure 19: Receptive field for convolutional layers with filter size 3 x 3 and stride = 2. Image courtesy of [18].

The receptive field can be calculated by iteratively moving through each layer of the model. In equation 20 it is shown how the receptive field can be calculated for each layer. R_k is the receptive field for layer k, f_k is the filter size for layer k and s_i is the stride for layer i.

$$R_k = R_{k-1} + \left((f_k - 1) * \prod_{i=1}^{k-1} s_i \right) \tag{20}$$

Dilated convolutions are another way to help increase the receptive field. The concept of dilated convolution can be explained by inserting spacing in the kernels. These spaces are set to zero. In figure 20 the concept of dilated filters are illustrated. Green elements are the true filter elements and the white elements are the inserted zero elements.

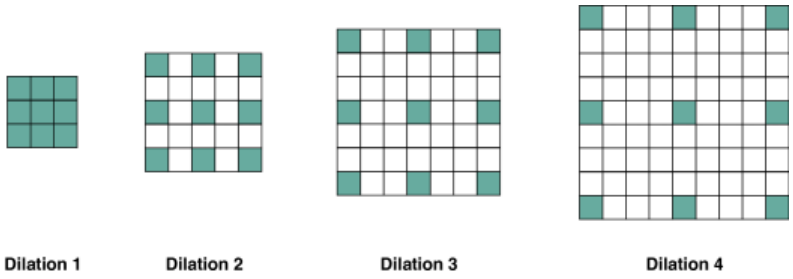


Figure 20: Filters with different dilation factors. Image courtesy of [10].

The hyperparameters of each convolutional layer in the model can be decided individually. In figure 21 the receptive field is shown with a growing dilation factor. In other words, the dilation factor increases for each layer in the model. It is often common to either use dilated convolutions or strided convolutions.

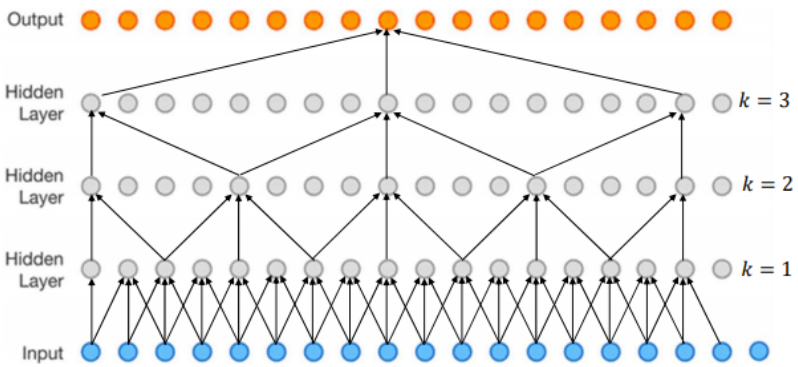


Figure 21: Receptive field with a growing dilation factor. Image courtesy of [18].

2.9.4 Learnable parameters

The learnable parameters of the network are numeric values that are updated during training. The learnable parameters for a CNN can be found in the convolutional layers, the fully connected layers and in the batch normalization layers. Max pooling layers only downsample its input and there are no learnable parameters present. ReLU layers are just a mathematical operation and there are no learnable parameters present.

Learnable parameters in convolutional layers: Convolutional layers convolve filters with its inputs. The element values in these filters are the learnable parameters of convolutional layers. Parameter sharing is used in the convolutional layers to control the number of learnable parameters. Parameter sharing assumes that it is useful to compute the same feature at different spatial locations in the input. This means that the same feature is selected at every height and width position of the input. This means that at each spatial location the same filter are used. The depth of the layer decides how many distinct filters that are found for each convolutional layer. The number of learnable parameters for a convolutional layer can be calculated according to equation 21, where w and h are the width and height of the filters, d_{in} and d_{out} are the input depth and output depth and n_{conv} are the number of parameters in the convolutional layer. The $+1$ term is the bias term of each filter.

$$n_{conv} = (w * h * d_{in} + 1) * d_{out} \tag{21}$$

Learnable parameters in fully connected layers: Fully connected layers usually have many more learnable parameters than convolutional layers. This is due to the fact that all nodes are connected together with corresponding weights. All input nodes have one distinct weight for every output node in the previous layer. The number of parameters for a fully connected layer can be calculated according to equation 22. Where n_{fc} is the number of learnable parameters for the fully connected layer and in and out are the number of inputs and outputs for the layer. The $+1$ term is the bias.

$$n_{fc} = (in + 1) * out \tag{22}$$

The number of learnable parameters for a network can be calculated by adding the number of learnable parameters for each layer.

2.9.5 Image pre-processing

As stated earlier one of the perks for CNNs and image classification problems is that pre-processing are almost non existing. The hidden layers of the network takes care of what features that are interesting, without human interaction during training. However, there are some pre-processing steps that are common to use such as zero centering and normalization. Zero centering subtracts the mean and centers the input data around 0. Normalizing the image means to squeeze the data to fit a specified range, e.g between -1 and 1 or 0 and 1. This is done to prevent different scales in data, but for image pixels which are normally between 0 and 255, normalizing the data is not strictly necessary.

2.9.6 Training the model

The model, or the architecture of the network can be considered as a shell which describes the learning environment of the network. This architecture is the static part of the network and describes the complexity and depth of the network in terms of number of parameters. These parameters are what the network learns and optimizes. For instance, the learnable parameters of a convolutional layer are the elements in the filters that convolves with the input to that layer. The process of learning these parameters are called training and will be achieved by backpropagation. These learnable parameters are also called weights and it will be referred to both interchangeably during this thesis. The process of training a model can be thought of as the dynamic part of the CNN where the weights of the static model are updated. Data flows in both direction of the model. In the following, the process of training a model will be explained.

Weight initialization: When creating a CNN and training it from scratch all of the weights and biases must be initialized to some value. The weights are initialized to small random numbers to help break symmetry. When initializing the weights randomly the nodes in the model will compute different outputs resulting in variations in the nodes. The weights should not be initialized equally because when all nodes computes the same output all nodes will get the same update and the nodes will all be similar. The biases are normally initialized to zero [6].

Softmax function: The softmax function normalizes the output from 0 to 1 where the sum of all outputs are 1. This yields a class score distribution of the outputs. This can be interpreted as a probability distribution. The softmax function is to be found at the last fully connected layer in the model where the number of nodes equals the number of classes. The softmax function is shown in equation 23. z is the ouput from the previous layer, $\sigma(z)_i$ denotes the class score of class number i and N is the number of classes. The sum of all $\sigma(z)_i$ is 1.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{n=1}^{N} e^{z_n}} \tag{23}$$

Loss function: The loss function measure the performance of a set of weights with the respective model. A forward pass is the dataflow from the input layer to the output. At the end of this forward pass a loss is calculated. This loss measures the inconsistency between the predicted labels and the true labels. One common loss function, especially for classification problems is the cross entropy loss. The cross entropy loss calculates the difference between two probability distributions. The network aims at minimizing the cross entropy loss, i.e reducing the distance between the true and predicted distributions. In a neural network it is common for the cross entropy loss to be computed

after the softmax function, since the cross entropy loss demands a probability distribution as input. This distribution is the class score distribution from the softmax function and it is worth mentioning that this is not actual probabilities, but predicted scores. In equation 24 the cross entropy loss is shown, where y is the true label and $\hat{y}$ is the predicted label.

$$E(y, \hat{y}) = -\sum_{i} y_i * log * \hat{y}_i \tag{24}$$

Backpropagation: Backpropagation is an algorithm which updates the weights of the model by applying an optimization function. After the loss have been calculated and the performance of the model is measured, it is time to update the weights, w in the model. The update of these weights are what is considered the training of the model. The calculated loss needs to be minimized which is done through backpropagation. Backpropagation moves in the opposite direction of the forward pass, and updates each weight on the way back from the output to the input. The weights are adjusted each iteration. The goal of the optimization function is to find a set of weights that minimizes the loss function. Stochastic gradient descent are maybe the most common optimization method for neural networks.
Stochastic gradient descent is a gradient descent algorithm which operates on single samples or mini-batches of data, hence stochastic. This method will later be referred to as SGD. The algorithm computes the gradient of the loss function with respect to the weights w and adjust each weight in the opposite direction of this gradient. In equation 25 it is shown how the weights in the network are updated. In equation 25 the weights are w, α (a hyperparameter) is the learning rate and *E(w)* is the loss at the i^{th} mini-batch with the current set of weights.

$$w = w - \alpha \nabla E_i(w) \tag{25}$$

It is possible to further develop equation 25, by including momentum. Stochastic gradient descent with momentum will further be referred to as SGDM. In the SGDM algorithm the change in weights are remembered at each iteration and the previous weight update will contribute to the next weight update. A physical interpretation of the momentum can be a ball rolling down a hill. The ball will gain velocity in steep parts. If the ball encounters a small valley the ball will roll over this valley because of the velocity the ball have gained. In equation 26 the momentum of the SGDM method is shown. Where δw can be considered the "velocity" of the ball in the example above. β can be thought of as a friction factor. In equation 27 the update rule for the SGDM method is shown, the new updated weights are a linear combination of the previous weights w and δw as shown in equation 26.

$$\delta w = \beta w - \alpha \nabla E_i(w) \tag{26}$$

$$w = w + \delta w \tag{27}$$

Overfitting: In statistics overfitting is defined as "The production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably" [24]. This definition is transferable to machine learning and deep learning. If the model classifies datasamples used in training very accurate, but classifies new unseen datasamples inaccurately, the model is overfitted. A model can also be underfitted, this means that the model struggle to classify both datasamples used in training and new unseen datasamples. To help visualize the concept of overfitting and underfitting, figure 22 and figure 23 are provided. These figures illustrates a two dimensional binary classification problem. Red and

blue data samples are the two different classes and the black line/curve is the decision boundary. This decision boundary is the mathematical function that separates the two classes. In figure 24 an example of a good decision boundary is provided.

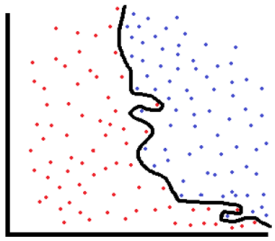


Figure 22: Overfitting: The decision boundary is too specific and fits the datasamples used in training perfectfly.

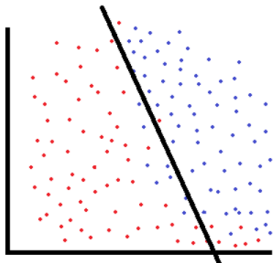


Figure 23: Underfitting: The decision boundary is too simple and the fit could be better.

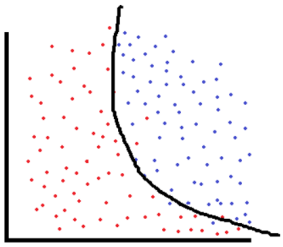


Figure 24: Good fit: The decision boundary fits the datasamples used during training well, and will still be a good boundary for classifying new unseen datasamples.

2.10 Transfer learning

Transfer learning is a technique for transferring knowledge from one problem over to another problem. Transfer learning is one of more techniques within deep learning that can help solve the challenge of a small dataset. From the human perspective it makes sense that the ability to recognize cars will help when trying to learn how to recognize motorcycles. Images of cars and motorcycles may share many features, such as wheels and that they are often located in the same environments such as garages and roads. Similar images share many obvious features, but also images that are not easily relatable share features. All natural images contain edges, corners and share similarities which the transfer learning technique exploits.
The process of collecting a wide variety in data and label every data sample can both be challenging and very time consuming. This often results in a classification problem with limited data. As stated above, transfer learning is a technique which can deal with such challenge.
There are a lot of pre-trained models available on the internet which can be downloaded and applied for the classification problem at hand. These models uses large datasets and often take a considerable amount of time to train even with a lot of computational resources. AlexNet is a well known network developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. This network was used to win the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The model was trained on ~1.2 million images with 1000 different classes originating from the ImageNet dataset. The model took 5-6 days to train on two GTX 580 3GB GPUs [19]. AlexNet is considered a deep network and is computationally expensive. GoogLeNet and SqueezeNet are other examples of pre-trained models that are also trained on the same dataset, but require less computational resources and memory.

2.10.1 Transfer learning as feature extractor

A pre-trained model contains many features that are applicable for the dataset at hand, and the need of adjusting these may be unnecessary. One approach towards transfer learning is to use the exact same features found using the original dataset which the model was trained on, and only decide which of these features that are important for the classes in the new dataset. During training, all the weights in the pre-trained model are frozen and the training only decides which features that are relevant for the new classes. The last layers which are bound to the classes in the original dataset are removed and replaced with layers that fit the number of current classes. For instance, in the AlexNet model, the last three layers shown in figure 25 will be replaced. The fully connected layer "fc8" in figure 25 will produce a 4096x1000 vector for each image that are sent through the model, because the original dataset which was used in training contained 1000 classes. This fully connected layer needs to be removed and replaced by a fully connected layer on the form 4096xD, where D is the number of classes [7]. A linear softmax classifier layer are added followed by a classification output layer which computes the cross entropy loss.

fc8 1000 fully connected layer	Fully Connected	1×1×1000	Weights 1000×4096 Bias 1000×1
prob softmax	Softmax	1×1×1000	-
output crossentropyex with 'tench'...	Classification Output	-	-

Figure 25: The last three layers in the AlexNet model

2.10.2 Transfer learning with fine tuning

Unlike the transfer learning method above, where all the weights in the pre-trained model were frozen, there is another approach where these weights can be adjusted. Transfer learning with fine tuning uses the new dataset to adjust the weights in the pre-trained network during training. When the training is finished, the features found with the original dataset is different from the new features. The new features are more specific to the dataset at hand. If this dataset is small and does not represent a general view of the samples that may occur in the classification problem, the new features could be prone to overfitting. Depending on the dataset it may be wise to use a small learning rate, such that the features found in the pre-trained model are not discarded so easily [7].
It is also possible to freeze some of the weights in the pre-trained model and fine tune a selection of the weights. Jason Yosinski et al. [32] explored how transferable features from one task is to another, i.e how suitable the features from the pre-trained model trained on the original dataset are for the new dataset at hand. The features found in the earlier convolutional layers are general features, like gabor filters and color blobs. The features get more specific to the classes in the new dataset used during training only later in the model. Early layer features can be used interchangeably between different datasets since these features are common for all images. The transferability gap grows when the distance between the tasks grows. This means that datasets that share similarities are more suitable for transferring features that originates from higher level features and vice versa.
Both the transferability of features and the size of the dataset are important factors when deciding which weights that should be fine tuned, if any.

3 Dataset

The dataset is the foundation for all of the image classification approaches and it is essential to design a good dataset. The success of a classifier is highly dependent on the dataset. The cap/no-cap dataset was manually created and labeled for this thesis. A wide variety of plastic bottles was gathered and recycled using a reverse vending machine located at Tomra. The RVM stored images captured from each delivery of plastic bottles. These images were uploaded to a computer in order to create the dataset.

3.1 Dataset features

When creating a dataset there are some key features that are important for an image classifier to perform good. For the different approaches of image classification the requirement of the dataset are somewhat the same, but each approach has some key features that are essential for the classification algorithm to succeed.

The **Pattern recognition** approach requires a big variety in the dataset. Plastic bottles and plastic bottle caps are unique and vary in size, shape and color. The plastic bottles also vary in location in the image. All of these features must be taken into account. A big variation in the dataset is needed such that as many examples as possible of different variations in the images are included.
For the **machine learning** approach it is important to have a dataset with many images. The machine learns each image quite specific and a big dataset and variety in data will help the machine generalize. This means that the machine need many images with variations to help distinguish between images of plastic bottles with the cap attached or detached that are not included in the dataset, i.e to be able to detect overfitting. The scale of the dataset is especially important for the **deep learning** approach.
The dataset must be manually created and this process is time consuming. A balance between the amount of data and time spent creating the dataset is of the essence. There are some techniques that can be applied for smaller datasets to help the performance of the classifier. Transfer learning and data augmentation are two common techniques.
Transfer learning are explained further in section 2.10, but the main concept is that images that originates from a different dataset share features with the dataset at hand. This makes it possible to transfer the learning made from another dataset to the problem at hand.
Data augmentation alters the data samples contained in the dataset at hand. These alterations can be rotations, translations, image cropping, image mirroring, adding Gaussian noise and more. By altering the data samples and adding these to the original dataset a wider variation of data is added. It is also common to perform data augmentation on the mini-batches of data samples during training.

3.2 Creating the dataset

The reverse vending machine has 6 cameras that are located in a circle around the entrance of the reverse vending machine. All these cameras capture images simultaneously of the plastic bottle when inserted. This yields a sequence of images from each delivery of plastic bottles with a view from 6 different angles. In figure 26, one delivery of one plastic bottle is visualized. This overview image is a high resolution image and contains all the pixels from all of the single images. The amount of images in one sequence depends on whether or not the plastic bottle is inserted with the top or bottom first.

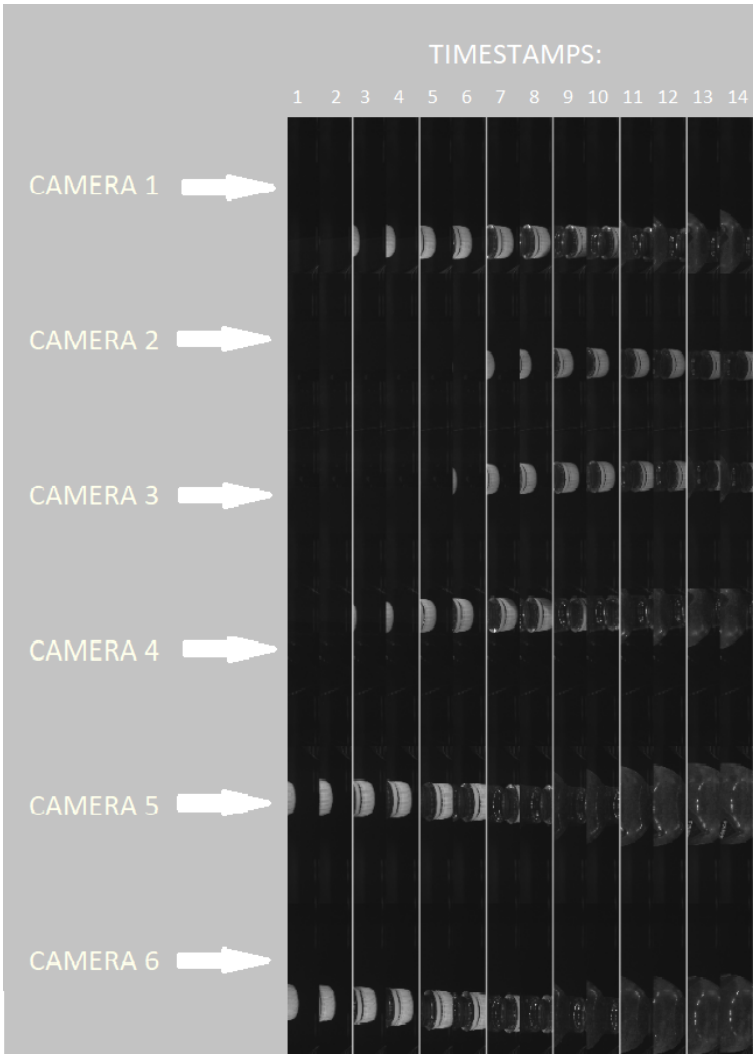


Figure 26: High resolution overview image of all single images of one bottle going through the reverse vending machine.

The overview image must be separated in single images as shown in figure 27. The image resolution of the single images are 1280x276 pixels. As seen in figure 27, the single images contains a lot of black areas where the plastic bottle is not located. These black areas are uninteresting and the next step is to find a region of interest for each of the single images. These region of interests keeps the spatial dimension in y-axis but reduces the spatial dimension in x-axis to 500 pixels. This is to ensure that every region of interest image have the same image resolution of 500x276 pixels. In figure 28 the region of interest image for the single image in figure 27 is shown. The region of interest was found by applying a canny edge detection filter and calculate the average value on the x-axis where the edges was located. The dataset used for this thesis is a labeled dataset that contains region of interest images as seen in figure 28.

Figure 27: One of the images in the overview image in figure 26.

Figure 28: The region of interest for the single image shown in figure 27.

The overview image contains several eligible single images. This results in a dataset that contains multiple ROI-images that originates from the same overview image, i.e the same plastic bottle. When randomly splitting the dataset into one training set and one test set the classification algorithm can be prone to overfitting. It is highly important to keep the training set and the test/validation set as separated as possible. Two separate datasets were manually created. One dataset was created as training set, and another dataset was created as test set. The plastic bottles used in these datasets were different plastic bottles. By manually creating these two datasets, it was made sure that the ROI-images in the two datasets do not originate from the same plastic bottle. The training data and the validation data should be as uncorrelated as possible so that the problem of overfitting is detectable by inspecting the training and validation loss, as well as the training and validation accuracy.

3.2.1 Dataset specifications

There are two datasets; The *training dataset* and the *validation/test dataset*. These two datasets both consists of ROI-images from different plastic bottles to make the datasets as uncorrelated as possible. The plastic bottles were also inserted into the RVM with both the top first and the bottom first. The images taken when inserting the plastic bottle with the top first are a bit different than the images taken when inserting the plastic bottle with the bottom first. Since there are 6 different cameras that captures images at different time steps and two different insertion directions, images that originates from the same plastic bottle can be said to be natural augmented. These different images are not augmented by a computer, nor augmented randomly. For each plastic bottle it exists images that are captured at different time steps, from twelve different angles (6 cameras and the bottle is inserted both with the top and bottom first).

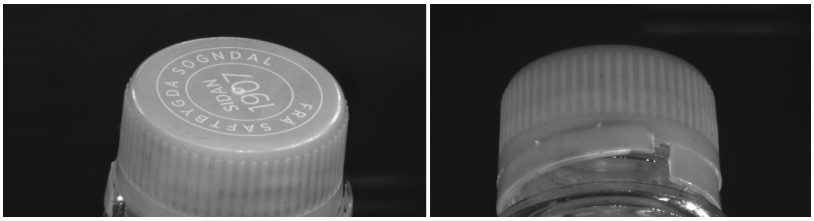


Figure 29: Difference in the images when inserting with the top and bottom first.

The dataset specifications are listed below:

Training dataset: The training dataset consists of 3068 ROI-images that originates from around 100 different plastic bottles. The same plastic bottles were used to capture images both when inserting with the top and bottom first. All images are grayscale images.

Validation dataset: The validation dataset consists of 1388 ROI-images that originates from around 40 different plastic bottles. The same plastic bottles were used to capture images both when inserting with the top and bottom first. All images are grayscale images.

Both datasets were gathered using the same RVM located at TOMRA headquarters.

4 Results and discussion

In this section the work produced in this thesis will be presented. The choices made for the different approaches will be discussed. The meaning of this thesis was to compare different image classification methods. In the following list, the different approaches are presented:

- Pattern recognition
  - Ellipses
  - Connected pixel intensity regions
  - Edge pixels on the whole image
- Machine learning
  - SVM
- Deep learning
  - CNN from scratch
  - Transfer learning

4.1 Pattern recognition approach

Since the features of the images must be manually extracted, the pattern recognition methods needs some thinking of what features that actually separates the two classes. It can be difficult to find one specific feature that manages to separate the two classes on its own. Therefore it can be wise to divide the classification problem into several classification algorithms which finds different features of the plastic bottle.
Some of the different features that catches the eye are listed below.

- Cap features
  - Vertical lines on the sides of the cap. These are made for grip when detaching the cap.
  - Uniform regions of pixel intensity values
- No cap features
  - Horizontal lines. These are made so that the cap can attach the bottle.
  - There are more ellipses or arcs present in the image when the cap is detached.

Even though there are features that separates the two classes, there are also features that both classes have in common. These shared features can make it tricky when trying to separate the two classes. The two classes, both have very similar shape and location in the image. There are also a big variety in the caps. Caps have different colors and some of the caps have the logo of its manufacturer or other printing on them. This makes it tricky to find one separate feature that can differentiate between the classes. In table 1 (section 4.1.4) the results of the pattern recognition methods are presented.

4.1.1 Ellipses

Ellipse detection was applied in hope to gather relevant information about the objects in the image. Ellipses in the image was found by deploying the ellipse detection algorithm written by Martin Simonovsky [13]. In figure 30, ellipses or arcs that are easily visible for the human eye are drawn for both classes. As seen in the figure, plastic bottles without caps have usually more ellipses present in the image than plastic bottles with the cap attached.

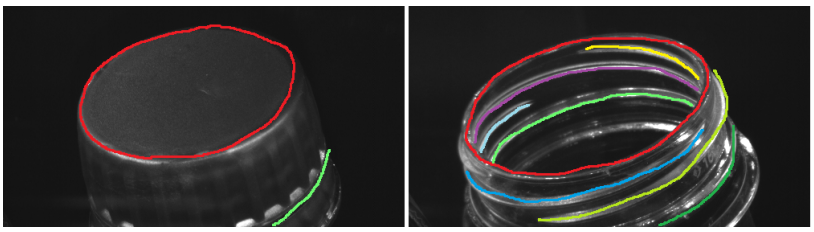


Figure 30: Ellipses drawn by hand. These ellipses or arcs are easily visible for the human eye.

The dataset contains some images where the top of the plastic bottle are located outside the image. It is important to have a general base of how the input images are presented to the ellipse detection algorithm. Therefore images where the plastic bottles are located in the top row of the image matrix were discarded. It was assumed that parts of the plastic bottle was outside the image if the top row in the image matrix contained edge pixels above a threshold. By discarding these images the dataset size is reduced, but since the dataset consists of multiple images at different time steps of the same plastic bottle, there are, in almost every case, still images of the same plastic bottle present in the dataset.

Overlapping ellipses: The number of ellipses detected in each image was set to 3. These 3 ellipses are the ellipses in the image with the highest score. Figure 31 and figure 32 each show 20 tiled images with the 3 highest scoring ellipses superimposed on the image in the color magenta.

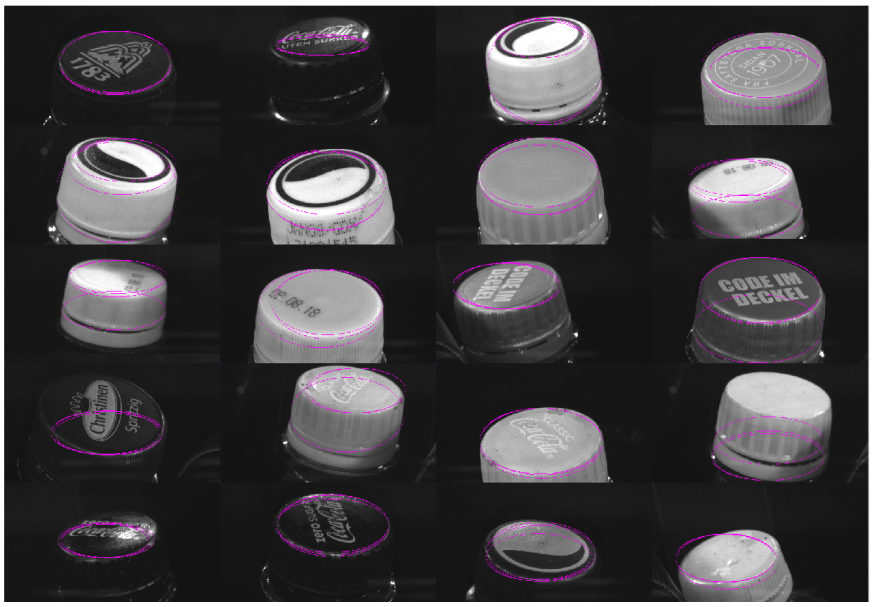


Figure 31: 20 different cap images with the 3 highest scoring ellipses superimposed.

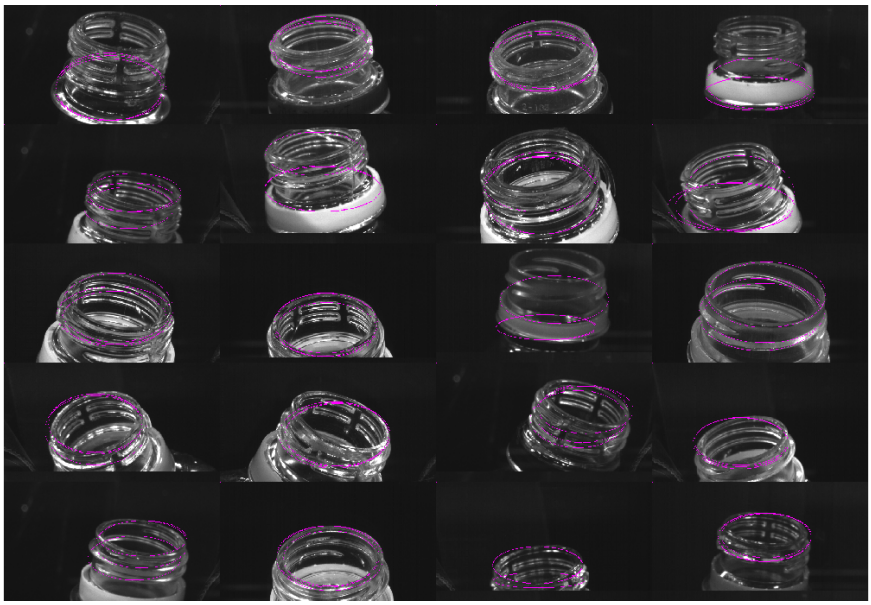


Figure 32: 20 different no-cap images with the 3 highest scoring ellipses superimposed.

The first idea behind the ellipse detection approach was to exploit the fact that cap images have fewer visible ellipses in the image than no-cap images and the ellipse detection algorithm would present more overlapping ellipses for the case of cap images, because there are fewer good ellipse fits in these images. Likewise the ellipse detection algorithm would present ellipses at different locations

for the no-cap images, due to the presence of several good ellipse fits. As it can be seen in figure 31 and 32, the ellipses detected do not follow the hypothesis. The ellipses located in the cap images do not consequently overlap.

Edges inside ellipse: For the case of cap images, at least one ellipse tend to be located around the top of the cap. The one ellipse with the highest score tend to be the ellipse that fits around the top of the cap. Since the ellipse detection algorithm detect the top of the cap quite consistent, this area can be extracted. These areas vary a lot from plastic bottle to plastic bottle. The top of the caps often have different printings such as date stamps, the manufacturers logo and so on.
But some caps have nothing printed on top of the cap. These caps may be detected by applying a canny edge detection filter on the images and look for edges inside the detected ellipse. For these caps, there will be few edges located at the top of the cap. These images must still be separated from the no-cap images. To be able to separate these, the no-cap images must have edges located inside the detected ellipse. In figure 33 there is shown 20 cap images with a canny filter applied. The one ellipse with the highest score is superimposed on the image. In figure 34 the same as in figure 33 is shown, but for no-cap images.

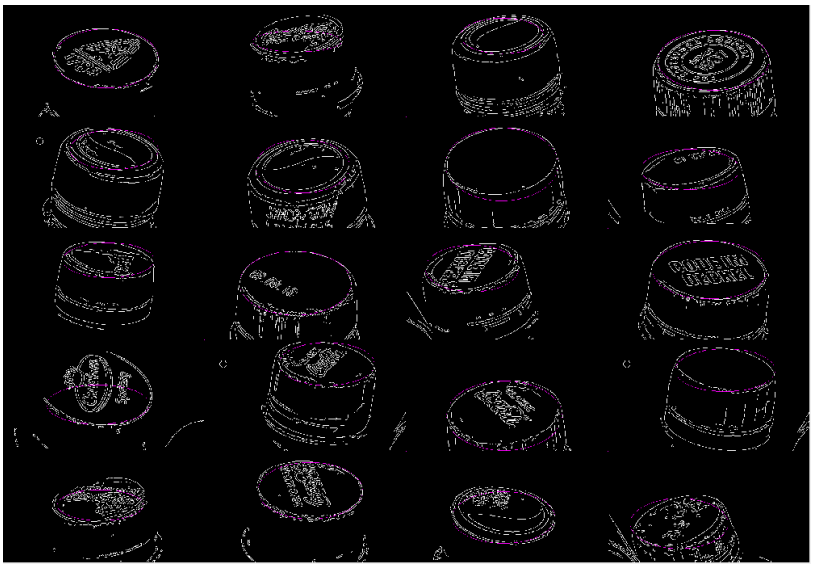


Figure 33: Canny cap images with the highest scoring ellipse superimposed.

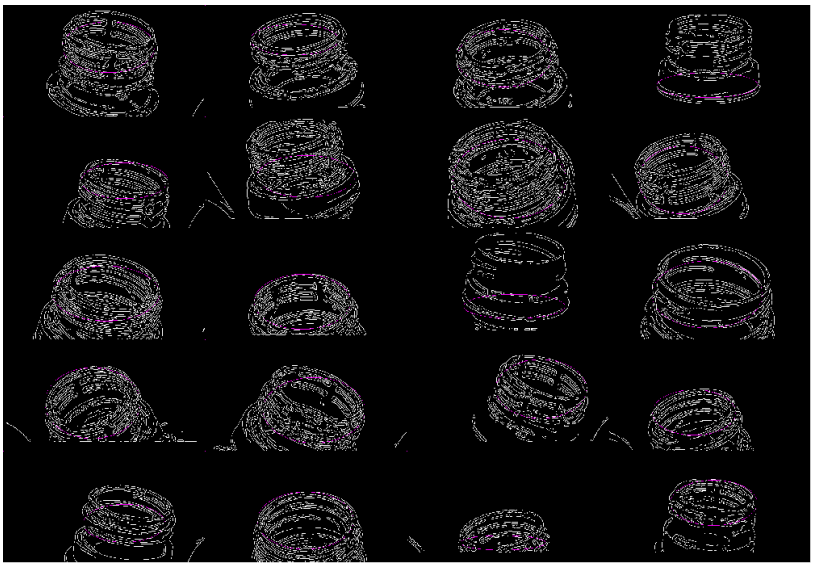


Figure 34: Canny no-cap images with the highest scoring ellipse superimposed.

The white pixels in the canny images represent edges. As it can be seen from figure 33 and 34 there are few white pixels inside the ellipse in some of the canny cap images. In almost every canny no-cap image there are white pixels present.
The number of white pixels inside the ellipse are counted. This gives a numerical measure of how many edges that are located inside the ellipse. The ellipses vary in size, and big ellipses will result in more white pixels on average than small ellipses. Therefore, the white pixel density inside the ellipse is found. This pixel density value is stored for each image as a numerical feature.
In figure 35 a plot visualizes the difference in edge density values inside the ellipses for the two classes. The red data samples are extracted from cap-images and the blue data samples are extracted from no-cap images. As it can be seen in the figure, the cap-images with no printing can not be separated by this approach. However, some of the no-cap images tend to have a larger edge pixel density inside the detected ellipse than the cap-images. There are 244 of the data samples that lie above the x-value of 0.15. If all these data samples are classified as no-cap images, there will be 4 misclassifications. This will result in an accuracy of 98.36% on 244 of the data samples. In table 1 (section 4.1.4) this result is listed along with the other pattern recognition results.

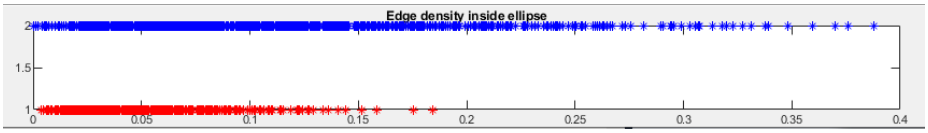


Figure 35: Plot of the edge density feature.

The ellipse detection algorithm is computationally heavy and time demanding. The classifiers that were made with the ellipse detection algorithm performed poorly. The features extracted from the images overlapped too much.

4.1.2 Connected pixel intensity regions

The idea behind this method was that plastic bottles with the cap attached have much larger regions that are connected with the same pixel intensity value. The cap is often represented with a uniform color and this method will manage to detect caps with a peak in pixel intensity values. In addition, the top of the cap is often flat and when capturing an image this will make the region reflect the same amount of light. When the cap is detached the tip of the bottle is shown. This tip reflects light differently due to the fact that the surface is not flat. This results in a more sparse area of pixel intensity values.
The process of this method and the results will be explained and presented under. A typical image of a plastic bottle with cap and a typical image of a plastic bottle without cap is used to present this method.

Image histogram: The image histogram is used to represent the pixel intensity values as a a plot. For plastic bottles with the cap attached there are peaks present in the image histogram at the pixel intensity values of the cap. For the plastic bottles without the cap attached the pixel intensity values are more sparse and the peaks present in the image histogram is not as significant. In figure 36 and figure 37 images with its corresponding image histograms are shown.

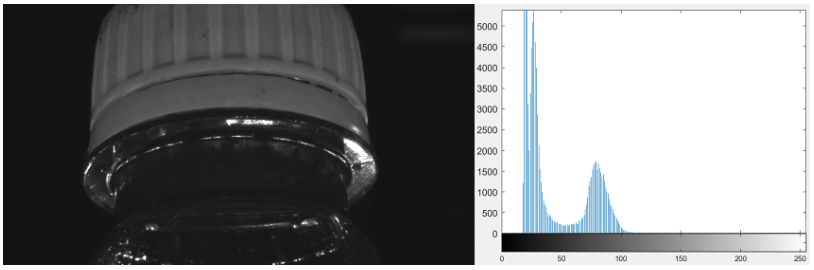


Figure 36: Cap - image histogram

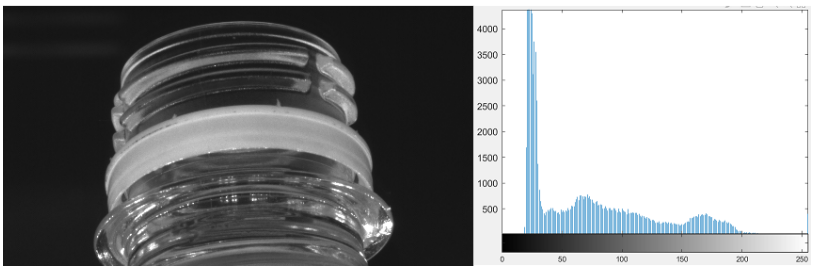


Figure 37: No cap - image histogram.

Peaks: The next step is to locate the peaks in the image histogram. The huge peak around 0 represents the black background in the image and is uninteresting, due to the fact that this background is present for either class. The interesting parts of the image histogram are the pixel intensity values after this large background peak. In figure 36 the pixel intensity values are quite dense which results in high narrow peaks. In figure 37 the pixel intensity values are quite sparse which results in low peaks with a bigger width.

There could be more than one peak in the image histograms and MATLABs *findpeaks* function locates peaks that are higher than 500 instances of the same pixel intensity value and wider than 5 pixel intensity values. Peaks that are located in near proximity are discarded. After these peaks are located, the pixel intensity value that are located at the top of the peak are stored.

Thresholding: After finding the pixel intensity value at the current peak, the image is thresholded. This thresholding process sets all pixels that are within 10% of the pixel intensity value at the peak to 255 (white), and all remaining pixels to 0 (black). In figure 38 the thresholded images of the two images in figure 36 and figure 37 are shown.

Figure 38: Threshold images, with cap attached (left) and with cap detached (right).

Connected regions: After the image have been thresholded connected pixel intensity value regions are found. MATLABs function *bwconncomp* is used to find regions of consecutive pixel intensity values. The function only looks at regions that are connected with pixel intensity value of 255 in the thresholded image. It is possible to require a connection in pixels with at least one in a neighborhood of either 4 or 8 pixels. The neighborhood kernel was set to 4 to prevent sparse data to connect to the same region. In figure 39 the concept of neighborhood is shown. Either one of the 4 most closest pixels must be connected to the center pixel or one of the 8 closest pixels must be connected to the center pixel.

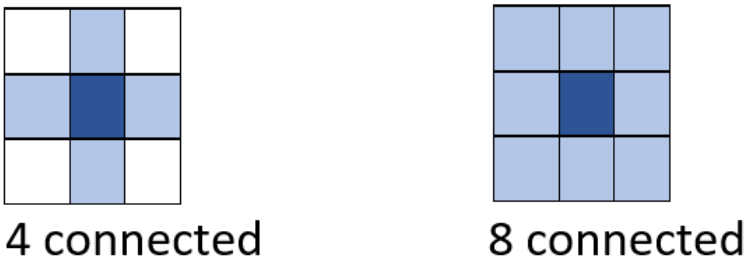


Figure 39: 4 connected kernel (left) and 8 connected kernel (right). Light blue color shows the accepted neighbor pixels.

MATLABs *bwconncomp* suggests connected regions. The region with the maximum area, i.e region with most pixels is extracted. This extracted region is the biggest consecutive region in the threshold image of pixel intensity value 255. For the two threshold images used in figure 38 the biggest region for each threshold image are represented as grey pixels in figure 40. The number of gray pixels in the leftmost image, cap image, are much bigger than the number of gray pixels in the rightmost image, no-cap image. The number of gray pixels is a numeric feature that can be exploited to separate between the two classes.

Figure 40: Biggest connected region (gray pixels) for the cap image (left) and biggest connected region (gray pixels) for the no-cap image (right). The remaining white pixels are pixels that are not connected to the biggest region.

By following the steps above, a dataset can be made which contains the number of pixels in the biggest connected region of the threshold image with the corresponding class label. In figure 41 a plot of this data is shown. Blue data points are labeled as cap and red data points are labeled as no-cap. The horizontal axis in the plot is on a logarithmic scale. It is worth mentioning that the presence of peaks in the image histogram is not always the case and these images are not included in the dataset. Another classifier must be applied for these images. The dataset contains 1542 data points out of 3068 images, i.e peaks were only present in 1542 of 3068 images.

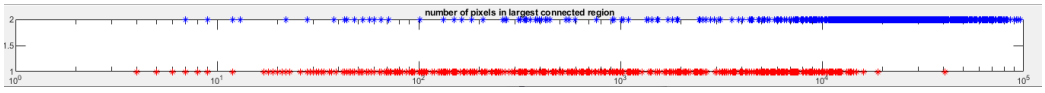


Figure 41: Plotted data samples, blue samples = cap, red samples = no cap. Horizontal axis have a logarithmic scale.

The data is not linearly separable, but can still provide some useful information. The blue data points tend to have a lot more connected pixels from the threshold image than the red data points. There are several ways to construct a classifier or a decision boundary. It is possible to set a decision boundary to some value and classify the regions with area bigger than the decision boundary as cap and regions with area smaller than the decision boundary as no-cap. This classifier will classify every data sample where a peak is found, but since the data is not linearly separable it will result in several misclassifications. Since it is of big importance to reach a high accuracy for this thesis another approach is to set two decision boundaries and classify images with regions above and below the two boundaries and keep all other instances unclassified. As shown in figure 41 the majority of data points above 15 000 are images with the cap attached. And there are majority of no-cap images below 1 000 on the horizontal axis. The accuracy of such a classifier will be better, on the cost of fewer classifications.
In the feature extraction routine, peaks were located in 1542 of the images in the training dataset. This means that the feature extraction routine discarded 1526 images. When classifying every data

sample that was not discarded during feature extraction, the best accuracy achieved was 0.8923 (89.23%). The decision boundary was set to 7 500. This means that all data samples with a connected region of over 7 500 pixels are classified as caps and all connected regions with less than 7 500 pixels are classified as no-caps. The classification time and feature extraction routine is, on average, 0.0098 seconds per data sample.
When introducing two decision boundaries and keeping the data samples between these two boundaries unclassified, the classifier performed better in terms of accuracy. By applying this method with an upper decision boundary of 11 000 and a lower decision boundary of 150, the accuracy are 0.9572 (95.72%). The number of classifications are 1051/3068, and the classification time are kept unchanged at 0.0098 per data sample on average.

4.1.3 Edge pixels on the whole image

The approaches above in section 4.1.1 and 4.1.2 can only be applied on a selection of the data, because the feature extraction routines discards data samples. A simple and computationally easy method is to apply the Canny edge detection algorithm on the input image, and then count how many edge-pixels that are present in the image. This approach is applicable to every image in the dataset. This means that the feature also can be used in the traditional machine learning approach. In figure 42 a plot of this data is shown. The data are not linearly separable, but it is possible to classify some of the data samples with good accuracy. As it can be seen in the plot, the red data points are the majority of data samples in the leftmost part of the plot and the blue data samples are in majority in the rightmost part of the plot. A decision boundary was set to 4 000 pixels. All data samples above the decision boundary was classified as no-caps and all data samples below the decision boundary was classified as caps. This resulted in an accuracy of 0.7627 (76.27%). The classification time averaged on 0.01 seconds per data sample, and all 3068 images in the dataset were classified.

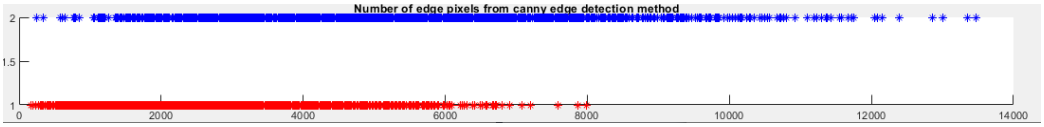


Figure 42: plotted data samples, red samples = cap, blue samples = no cap.

4.1.4 The different pattern recognition approaches

The pattern recognition approaches applied in this study struggles to classify all of the data samples with high accuracy. A portion of the data samples can be classified quite accurately in the different methods. Some of the methods discards input images during the feature extraction routine. These methods are not applicable for every data sample.

Pattern recognition approaches			
Name of classifier	Accuracy	Number of classifications	Classification time*
Pixel density - ellipse	0.9836	244/3068	1.6 seconds
Connected regions 1	0.8923	1542/3068	0.0098 seconds
Connected regions 2	0.9572	1051/3068	0.0098 seconds
Edge pixels - total image	0.7627	3068/3068 (all)	0.01 seconds

Table 1: Table with results of the pattern recognition approaches. *Average classification time per data sample.

As it can be seen in table 1, the approaches applied in this section are not very good. It seems like the features found in the images are able to separate some of the data samples very good, but only a small fraction of them. By combining multiple pattern recognition approaches that classifies a small portion of the dataset with high accuracy each method can separate some of the data samples, and when the number of different classifiers gets larger the chance of covering more of the data samples gets bigger.
It is also possible to have multiple poor classifiers. Each of these votes for a class. By combining all of these votes, the majority wins. Random Forest is such a method [26], where each classifier is represented as a decision tree. It is important to not have classifiers that operates on correlated data. This will increase the error rate. Since each classifier has one vote, features that are correlated will have "more" than one vote.
Multiple classifiers may increase accuracy, but it is also worth mentioning that classification time will increase as the number of classifiers increase. Since the classification algorithm shall run in real time, it is important to not only consider accuracy, but also classification time.

4.2 Traditional machine learning

The traditional machine learning techniques requires the user to manually extract features. It can be challenging to point out features that represents each class, but it can be even more challenging to find a method for extracting these features and give them a numeric value. In section 4.1 different pattern recognition approaches were applied. These approaches found one feature and tried to classify a selection of images by this particular feature. In this section about traditional machine learning, manually extracted features are combined. This means that there are several different features stored in a dataset. This dataset can be of different dimensions, i.e a different amount of features can be used. One feature represent one dimension and by combining a lot of features the feature space grow and can be hard to visualize in a plot.

4.2.1 Manually extracted features

The biggest challenge for this approach is to extract good features. Good features are features that help separate the two classes. A good feature is also a feature that is computationally cheap and fast to extract. In section 4.1, different approaches were applied and some features were only applicable to a subset of the dataset. It is important to find features that can be extracted for every image in the dataset. Finding good features can be very time consuming. Three features that is easily found for every image in the dataset is listed below:

- Number of edge pixels when applying a horizontal prewitt filter
- Number of edge pixels when applying a vertical prewitt filter
- Number of edge pixels when applying the Canny edge detection method

These features represents a 3 dimensional feature space. All of the features are based on edge detection and are somewhat correlated. The idea of detecting both horizontal edge pixels and vertical edge pixels is based on the list presented in the beginning of section 4.1. This feature space is visualized in figure 43. As it can be seen in figure 43, the feature space is not linearly separable.

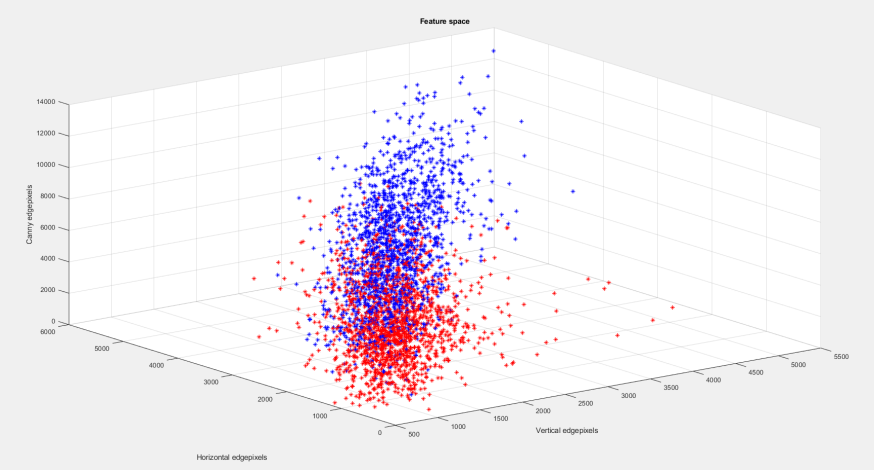


Figure 43: Scatter plot of the 3 dimensional feature space. Red points represents cap images and blue points represents no-cap images.

This means that there is no linear plane that can fully separate the two classes. In addition to the features listed above, the pixel intensity of the biggest connected region and the size (number of pixels) of this region were extracted. In section 4.1.2 the biggest connected regions was extracted by finding peaks in the image histograms. Peaks that met the critera that were set were not present in every image histogram, this made the method applicable to only a subset of the dataset. To be able to find the biggest region in every image, a more time consuming approach was applied. For every pixel intensity value above 35 (less than 35 is the black background) the biggest connected region was found. Of all these regions, the number of pixel and the pixel intensity of the biggest connected region was stored.

By combining these features, a five dimensional dataset was made. Different combinations of the features were used to train the classifier.

4.2.2 Support vector machine (SVM)

SVM was applied with the data sample features extracted in section 4.2.1. The function *fitcsvm* provided by MATLAB is used for one- or two-class classification problems with a low dimensional feature space. This function maps the input data using kernel functions. The kernel function used was the radial basis function (RBF). The data samples were normalized. The normalization routine both zero-centers and scales the data. The SVM model classifies data samples very quickly, but it is important to remember that the features from the images must be extracted for each classification which can be time consuming. When a plastic bottle are inserted into the RVM and images are captured, feature extraction must be able to be applied on every one of these images.

When combining all five extracted features the validation accuracy was 0.5540 (55.40%) and the training accuracy was 0.7754 (77.54%). The average classification time was 0.2122 seconds. This classifier is clearly overfitted, because it fits the training samples a lot better than the test samples. Poor features can be considered as noise to the model. When a model tries to create a decision boundary based on noise, there is a big chance of overfitting. It is important to carefully select features, such that poor features are detected and can be discarded. It is also worth mentioning that the classification time with feature extraction increases as more features are added.

The best result achieved with SVM was a classifier which used the combination of features as in figure 43. The SVM model classified 1388 test images with a validation accuracy of 0.755 (75.5%). The training accuracy was 0.7751 (77.51%). Since the training dataset and test dataset are kept separated the classifier does not look overfitted. The average classification time with feature extraction applied on all test images was 0.0137 seconds.

The SVM approach turned out to perform quite similar as to the approach in section 4.1.3 (edge pixels on the whole image). Both these approaches were based on edge features. The accuracy for the classifiers are poor.
All of the remaining methods are based on deep learning and will be presented in section 4.3.

4.3 Deep learning approach

Convolutional neural networks have proven to be good at image classification problems. The same turned out to apply for this thesis as well. A challenge with CNNs are memory, computation and scale of the dataset. Since the image classifier application is going to run on an embedded system in real time, the networks must be small and fast. Both networks trained from scratch and transfer learning networks are presented in this section.

4.3.1 Training and validation set

When training a classifier the dataset is split into a training set and a validation set. The dataset created contains several images of the same plastic bottle from different angles due to the six cameras, and from different translations due to the different time steps. MATLAB have a function that randomly split the dataset into one training and one validation set. But since some of the data samples in the dataset are more similar to one another, due to the fact that they are the exact same plastic bottle, but from a different view, it is important to keep these data samples in the same set. If these data samples appears both in the training set and in the validation set, the results may be unreliable. By randomly splitting the dataset into a training and validation set the data samples that originates from the same plastic bottle will be mixed. To prevent this from happening, two datasets were manually created; one for training and one for validation. These two datasets where created with different plastic bottles to make sure the results presented in this thesis are reliable.

4.3.2 CNN - from scratch

Creating CNNs from scratch are rarely recommended for image classification tasks where only small amount of data is accessible. In such cases it is more common to use a pre-trained model either as weight initialization or as a pure feature extractor [7]. The upside of creating a CNN model and train it from scratch is that one is independent of other architectures and are able to fully decide on the design of the model.
In this section, results from different trained models will be presented. Training plots and architecture schemes will be provided and discussed.
The filters that are used in the convolutional layers in the models are all of size 3 x 3. 3 x 3 filters yields a larger receptive field per parameter than larger filters. By following equation 21 the number of parameters for convolutional layers can be calculated, and by following equation 20 the receptive field for the convolutional layers can be calculated. These equations shows that the receptive field for two convolutional layers with filter sizes 3 x 3 is 5. This is the same receptive field as for one convolutional layer with 5 x 5 filters. The number of learnable parameters for the case of two convolutional layers with 3 x 3 filters, where the number of filters in each layer is C, is $20C^2$. For the case of one convolutional layer with 5 x 5 filters with the same number of filters at each layer the number of learnable parameters are $26C^2$. Therefore, 3 x 3 filters were chosen for the convolutional layers.
Since the classification application will be implemented on an embedded system for real-time classifications, it is of high importance to have an application which is accurate, runs fast and require small amount of memory.

In table 2, several CNNs trained from scratch are presented. Each model is named in the leftmost column in the table. This name will be used to refer to these models later in the text. It is worth mentioning that the deep learning approaches classifies every single image in the dataset.

CNNs trained from scratch				
Name	Test accuracy	Size	Learnable parameters	Classification time*
Conv_1	0.9935	6 465 kB	4 313 090 370	0.0148 s
Conv_2	0.9849	1 882 kB	398 370	0.0066 s
Conv_3	0.9777	1 512 kB	296 514	0.0062 s
Conv_4	0.9589	492 kB	16 178	0.0014 s
Conv_5	0.9416	450 kB	4 866	0.0012 s
Conv_6	0.8192	439 kB	1 906	0.0011 s
Conv_7	0.7774	432 kB	318	0.00097 s

Table 2: Table of the results of CNNs trained from scratch. *Average time per data sample.

The classification time for the different networks in table 2 may vary a bit from each time they are executed. These measured classification times can not be relied on when applying the networks on a different platform with different hardware. But since all of these times where measured on the same computer it gives a good indication of the classification time relative to each other. The networks in table 2 were executed using one single 8 th Gen intel I7-8650U CPU.

All of the network models from table 2 has taken inspiration from the AlexNet model. In the earlier layers, the networks are built up by convolutional layers followed by batch normalization, ReLU activation functions and max pooling. In the latter layers there are fully connected layers followed by a ReLU or softmax function. The networks in the table are quite similar when it comes to the idea of the architecture, but the networks vary in the amount of the different layers and in the depth of the network, i.e how many filters that are found at each convolutional layer. After a quick look at the table it is clear that the accuracy of the networks are highest for the networks with the highest number of learnable parameters. When the number of learnable parameters decreases the accuracy gets worse. This is the same observation found by Alex Krizhevsky et. al [19]. In figure 44 it can be seen how the accuracy increases as the number of learnable parameters increases. The seven orange scatter points in the plot represents the seven different CNNs from table 2.

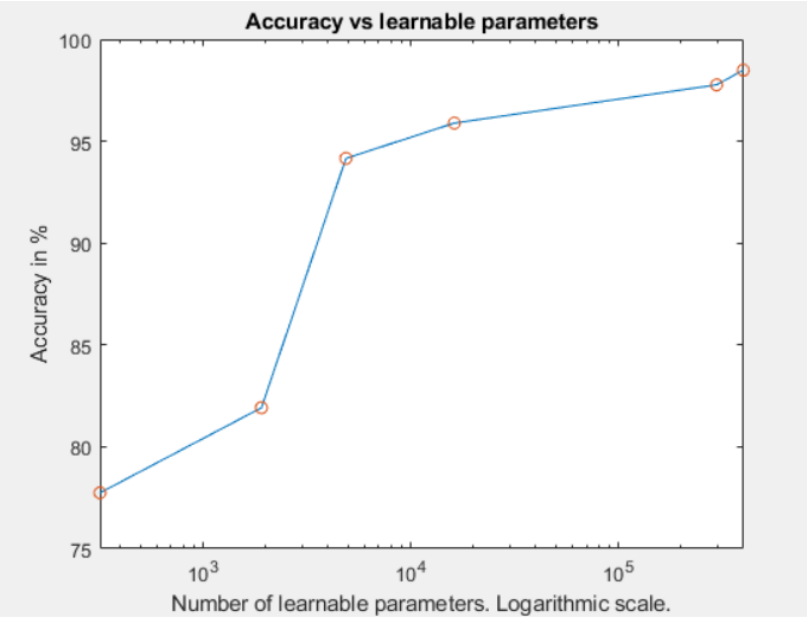


Figure 44: Validation accuracy vs learnable parameters.

All seven networks presented in table 2 was trained with the same training options. The training options used for the networks are listed below.

- Training options
  - Optimization method: Stochastic gradient descent with momentum (SGDM).
  - Learning rate: Constant learning rate of 0.003
  - Mini-batch size: 128.
  - Number of epochs: 40.

One epoch is when all the data samples in the dataset has been seen one time by the network. The number of epochs decides for how long the training is executed and was set to 40. The mini-batch size is how many data samples that flows through the network at each iteration during training. A small mini-batch size gives quicker convergence, but will introduce noise as a small mini-batch does not represent a wide variety of data samples. The mini-batch size was set to 128 as this gave a good balance between a relatively quick convergence and a good end result in terms of accuracy.

In the following pages, training plots with corresponding network architectures for the CNNs in table 2 are presented. The names of the networks in the leftmost column in the table will be used to refer to the different CNNs. Figures of the training plots and the architectures will all be presented before any discussion takes place. The training options are similar for each network and the general layout of the architectures are similar.

ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES	TOTAL LEARNABLES
1	imageinput 276x500x1 images with 'zerocenter' normalization	Image Input	276×500×1	-	0
2	conv_1 32 3x3x1 convolutions with stride [2 2] and padding 'same'	Convolution	138×250×32	Weights 3×3×1×32 Bias 1×1×32	320
3	batchnorm_1 Batch normalization with 32 channels	Batch Normalization	138×250×32	Offset 1×1×32 Scale 1×1×32	64
4	relu_1 ReLU	ReLU	138×250×32	-	0
5	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	69×125×32	-	0
6	conv_2 64 3x3x32 convolutions with stride [2 2] and padding 'same'	Convolution	35×63×64	Weights 3×3×32×64 Bias 1×1×64	18496
7	batchnorm_2 Batch normalization with 64 channels	Batch Normalization	35×63×64	Offset 1×1×64 Scale 1×1×64	128
8	relu_2 ReLU	ReLU	35×63×64	-	0
9	conv_3 64 3x3x64 convolutions with stride [2 2] and padding 'same'	Convolution	18×32×64	Weights 3×3×64×64 Bias 1×1×64	36928
10	batchnorm_3 Batch normalization with 64 channels	Batch Normalization	18×32×64	Offset 1×1×64 Scale 1×1×64	128
11	relu_3 ReLU	ReLU	18×32×64	-	0
12	conv_4 128 3x3x64 convolutions with stride [1 1] and padding 'same'	Convolution	18×32×128	Weights 3×3×64×128 Bias 1×1×128	73856
13	batchnorm_4 Batch normalization with 128 channels	Batch Normalization	18×32×128	Offset 1×1×128 Scale 1×1×128	256
14	relu_4 ReLU	ReLU	18×32×128	-	0
15	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	9×16×128	-	0
16	conv_5 128 3x3x128 convolutions with stride [1 1] and padding 'same'	Convolution	9×16×128	Weights 3×3×128×128 Bias 1×1×128	147584
17	batchnorm_5 Batch normalization with 128 channels	Batch Normalization	9×16×128	Offset 1×1×128 Scale 1×1×128	256
18	relu_5 ReLU	ReLU	9×16×128	-	0
19	conv_6 256 3x3x128 convolutions with stride [1 1] and padding 'same'	Convolution	9×16×256	Weights 3×3×128×256 Bias 1×1×256	295168
20	batchnorm_6 Batch normalization with 256 channels	Batch Normalization	9×16×256	Offset 1×1×256 Scale 1×1×256	512
21	relu_6 ReLU	ReLU	9×16×256	-	0
22	conv_7 256 3x3x256 convolutions with stride [1 1] and padding 'same'	Convolution	9×16×256	Weights 3×3×256×256 Bias 1×1×256	590080
23	batchnorm_7 Batch normalization with 256 channels	Batch Normalization	9×16×256	Offset 1×1×256 Scale 1×1×256	512
24	relu_7 ReLU	ReLU	9×16×256	-	0
25	maxpool_3 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	4×8×256	-	0
26	conv_8 128 3x3x256 convolutions with stride [2 2] and padding 'same'	Convolution	2×4×128	Weights 3×3×256×128 Bias 1×1×128	295040
27	batchnorm_8 Batch normalization with 128 channels	Batch Normalization	2×4×128	Offset 1×1×128 Scale 1×1×128	256
28	relu_8 ReLU	ReLU	2×4×128	-	0
29	maxpool_4 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	1×2×128	-	0
30	fc_1 512 fully connected layer	Fully Connected	1×1×512	Weights 512×256 Bias 512×1	131584
31	relu_9 ReLU	ReLU	1×1×512	-	0
32	fc_2 128 fully connected layer	Fully Connected	1×1×128	Weights 128×512 Bias 128×1	65664
33	relu_10 ReLU	ReLU	1×1×128	-	0
34	fc_3 2 fully connected layer	Fully Connected	1×1×2	Weights 2×128 Bias 2×1	258
35	softmax softmax	Softmax	1×1×2	-	0
36	classoutput crossentropyex with classes 'cap' and 'no cap'	Classification Output	-	-	0

Figure 45: Architecture for *Conv*_1.

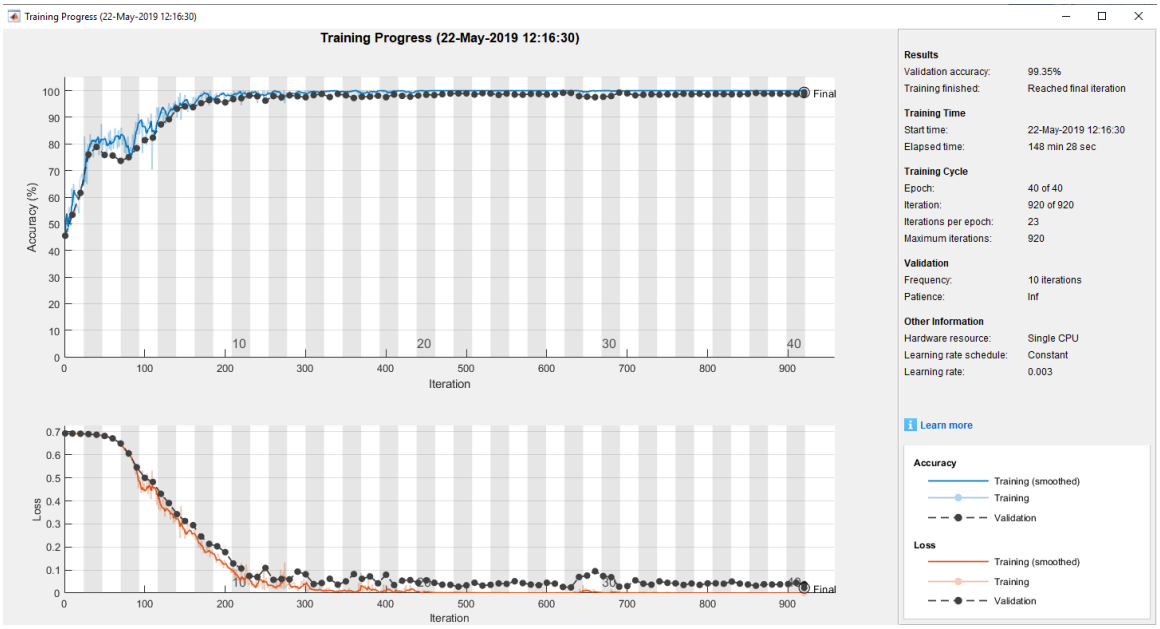
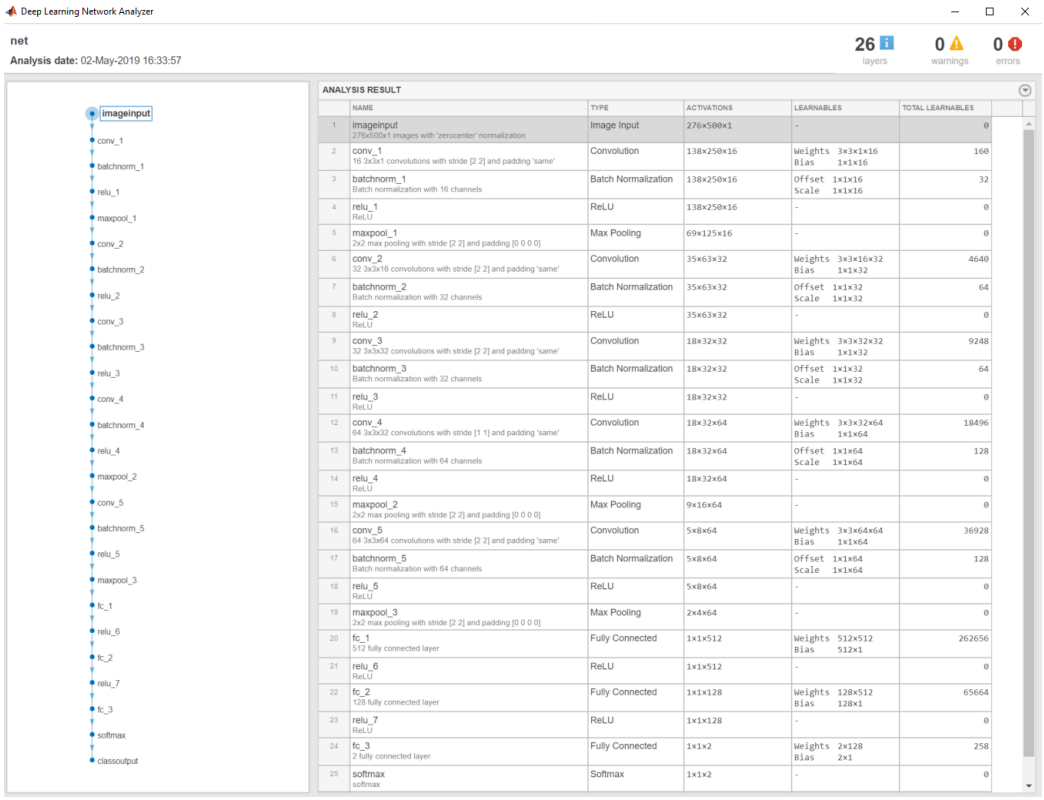


Figure 46: Training plot for $Conv\_1$.



	NAME	TYPE	ACTIVATIONS	LEARNABLES	TOTAL LEARNABLES
1	imageinput 276x500x1 images with 'zerocenter' normalization	Image Input	276×500×1	-	0
2	conv_1 16 3x3x1 convolutions with stride [2 2] and padding 'same'	Convolution	138×250×16	Weights 3×3×1×16 Bias 1×1×16	160
3	batchnorm_1 Batch normalization with 16 channels	Batch Normalization	138×250×16	Offset 1×1×16 Scale 1×1×16	32
4	relu_1 ReLU	ReLU	138×250×16	-	0
5	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	69×125×16	-	0
6	conv_2 32 3x3x16 convolutions with stride [2 2] and padding 'same'	Convolution	35×63×32	Weights 3×3×16×32 Bias 1×1×32	4640
7	batchnorm_2 Batch normalization with 32 channels	Batch Normalization	35×63×32	Offset 1×1×32 Scale 1×1×32	64
8	relu_2 ReLU	ReLU	35×63×32	-	0
9	conv_3 32 3x3x32 convolutions with stride [2 2] and padding 'same'	Convolution	18×32×32	Weights 3×3×32×32 Bias 1×1×32	9248
10	batchnorm_3 Batch normalization with 32 channels	Batch Normalization	18×32×32	Offset 1×1×32 Scale 1×1×32	64
11	relu_3 ReLU	ReLU	18×32×32	-	0
12	conv_4 64 3x3x32 convolutions with stride [1 1] and padding 'same'	Convolution	18×32×64	Weights 3×3×32×64 Bias 1×1×64	18496
13	batchnorm_4 Batch normalization with 64 channels	Batch Normalization	18×32×64	Offset 1×1×64 Scale 1×1×64	128
14	relu_4 ReLU	ReLU	18×32×64	-	0
15	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	9×16×64	-	0
16	conv_5 64 3x3x64 convolutions with stride [2 2] and padding 'same'	Convolution	5×8×64	Weights 3×3×64×64 Bias 1×1×64	36928
17	batchnorm_5 Batch normalization with 64 channels	Batch Normalization	5×8×64	Offset 1×1×64 Scale 1×1×64	128
18	relu_5 ReLU	ReLU	5×8×64	-	0
19	maxpool_3 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	2×4×64	-	0
20	fc_1 512 fully connected layer	Fully Connected	1×1×512	Weights 512×512 Bias 512×1	262656
21	relu_6 ReLU	ReLU	1×1×512	-	0
22	fc_2 128 fully connected layer	Fully Connected	1×1×128	Weights 128×512 Bias 128×1	65664
23	relu_7 ReLU	ReLU	1×1×128	-	0
24	fc_3 2 fully connected layer	Fully Connected	1×1×2	Weights 2×128 Bias 2×1	258
25	softmax softmax	Softmax	1×1×2	-	0

Figure 47: Architecture for *Conv_2*.

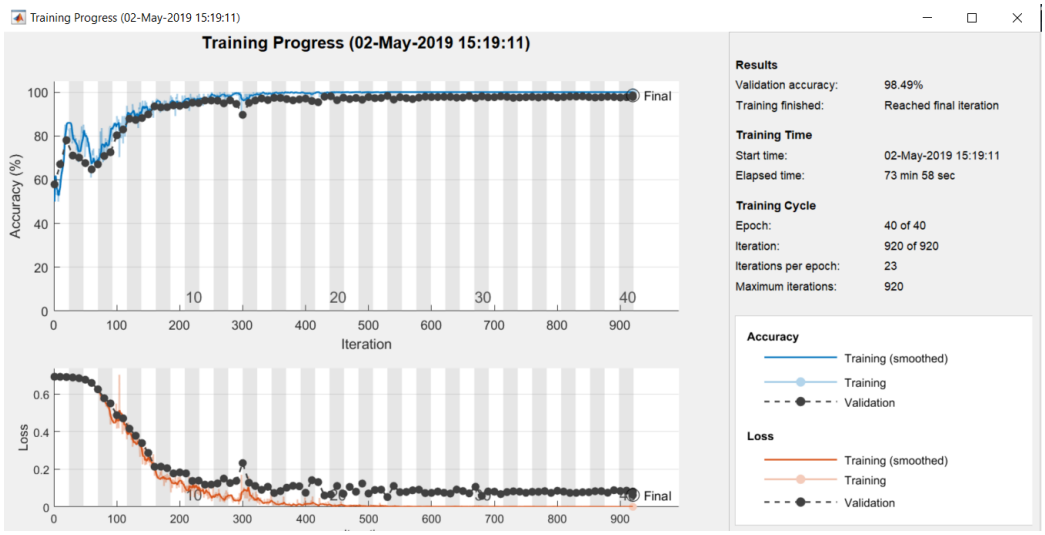


Figure 48: Training plot for *Conv_2*.

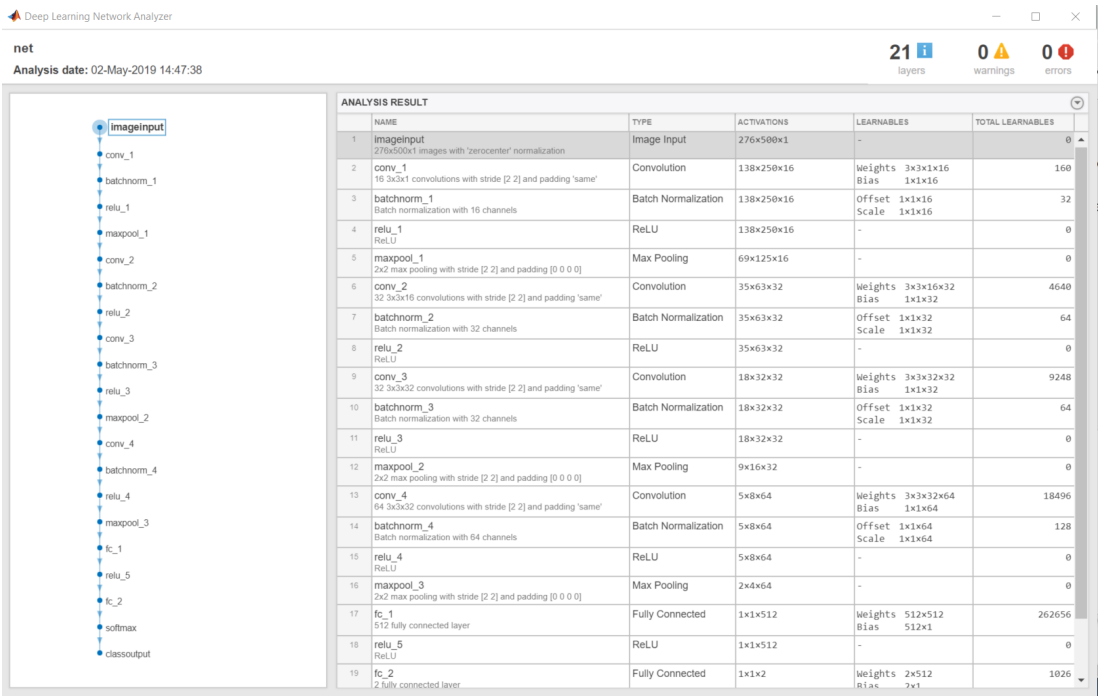


Figure 49: Architecture for *Conv_3*.

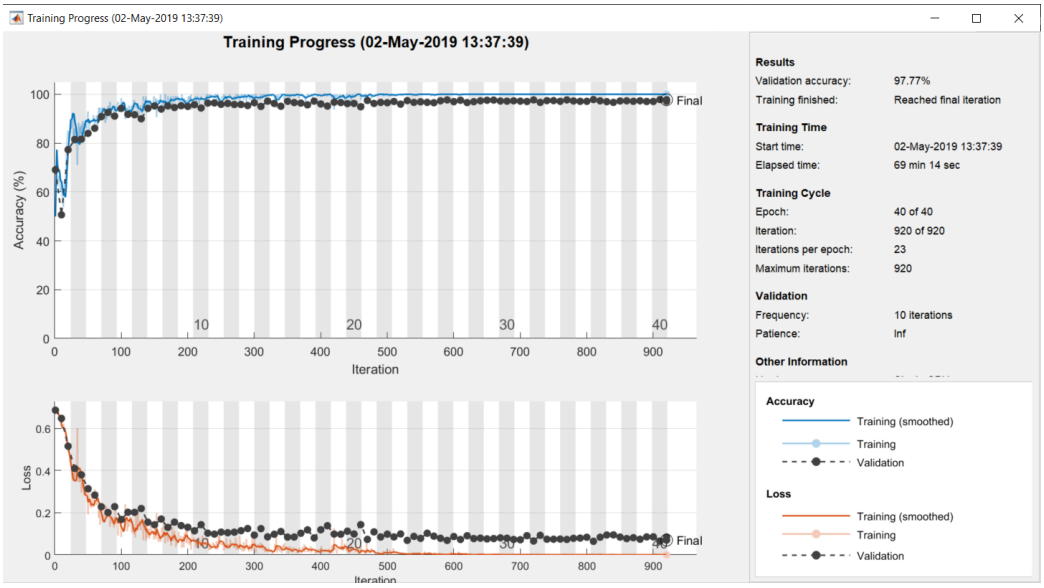


Figure 50: Training plot for *Conv_3*.

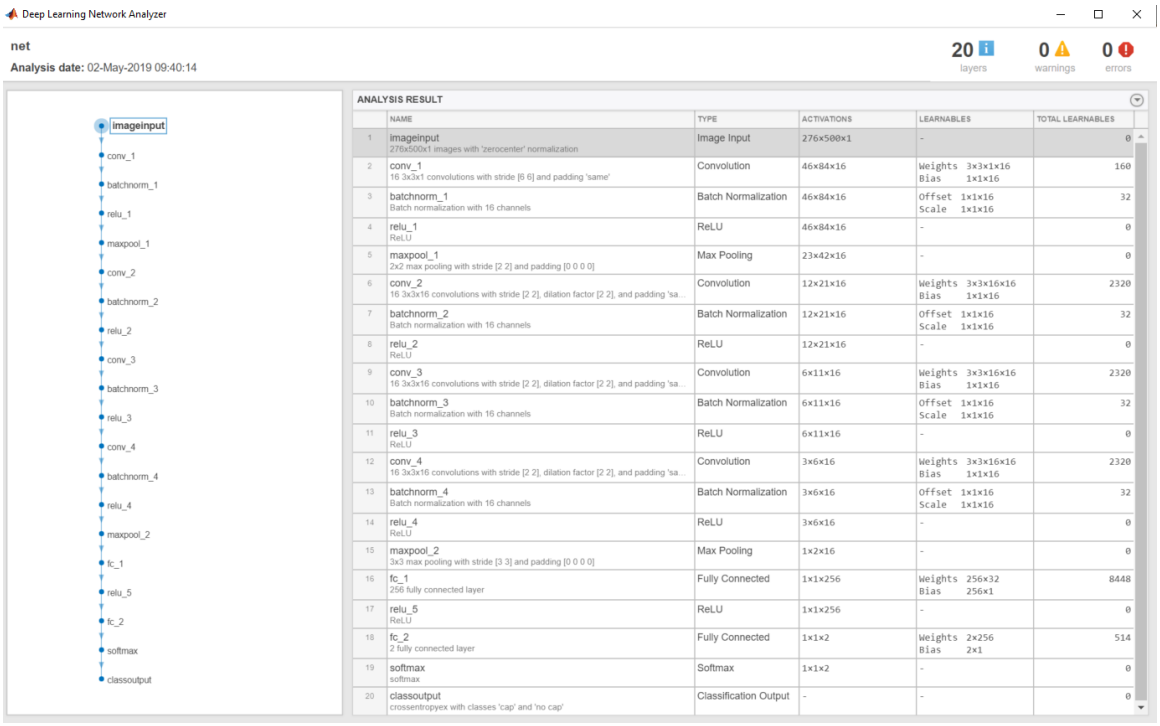


Figure 51: Architecture for *Conv_4*.

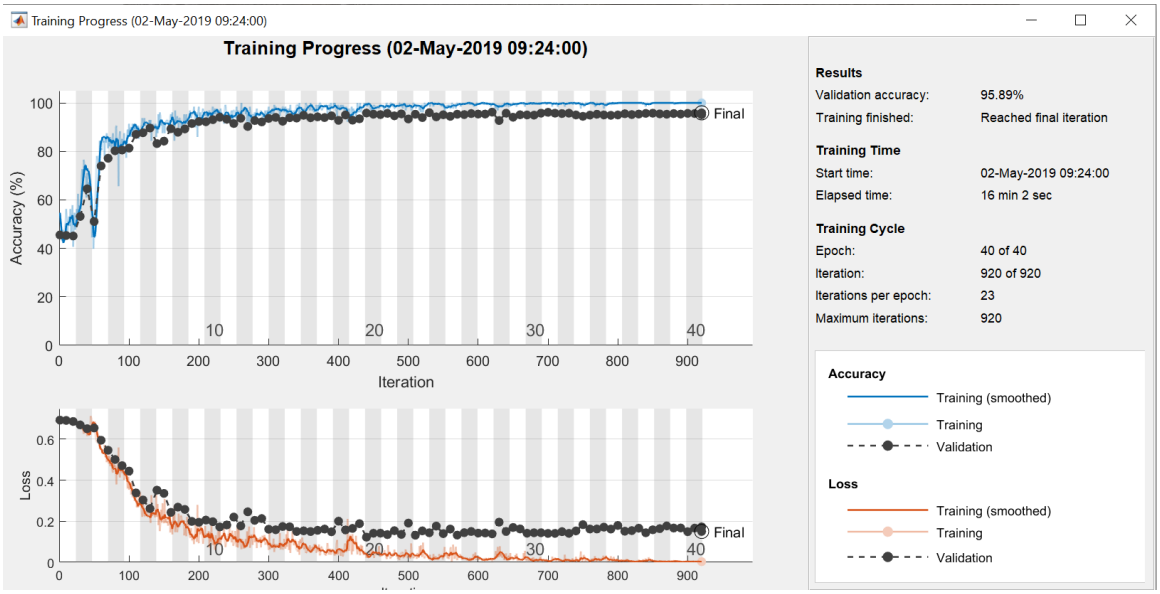


Figure 52: Training plot for *Conv_4*.

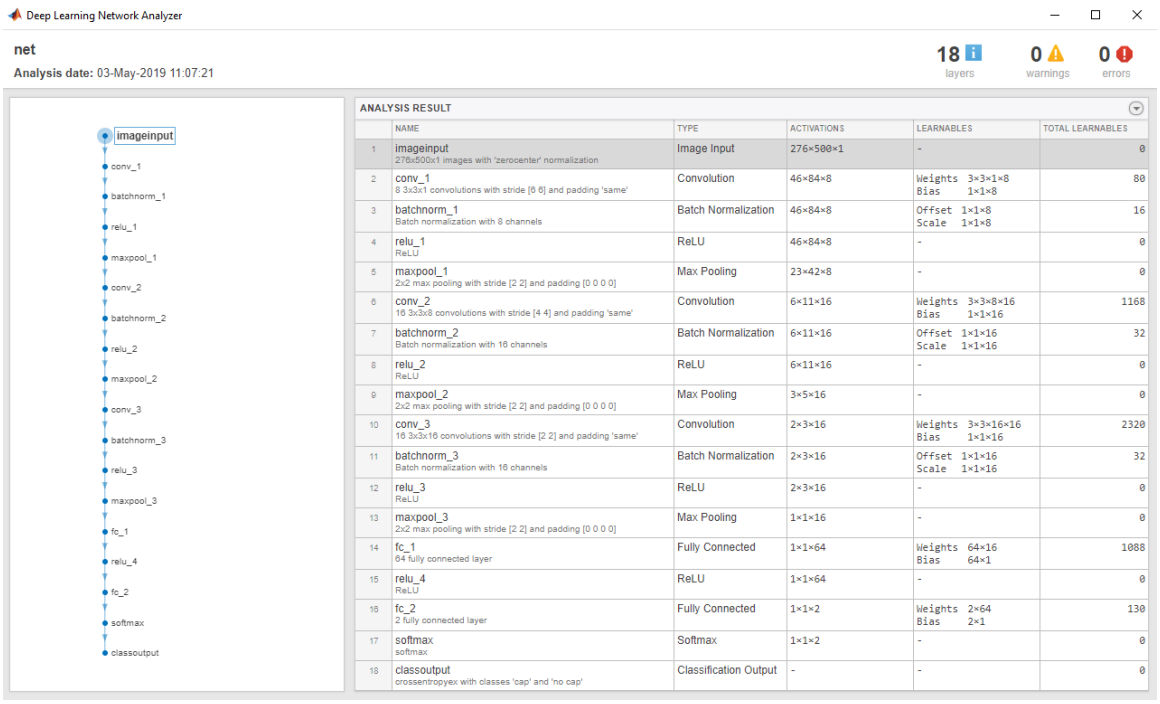


Figure 53: Architecture for *Conv_5*.

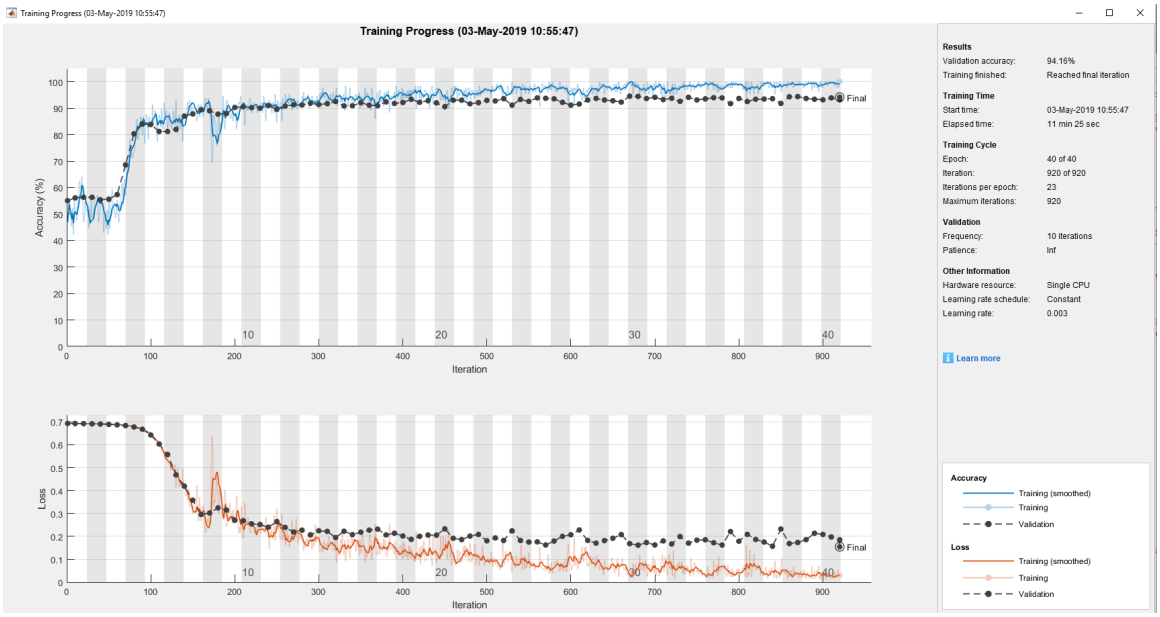


Figure 54: Training plot for *Conv_5*.

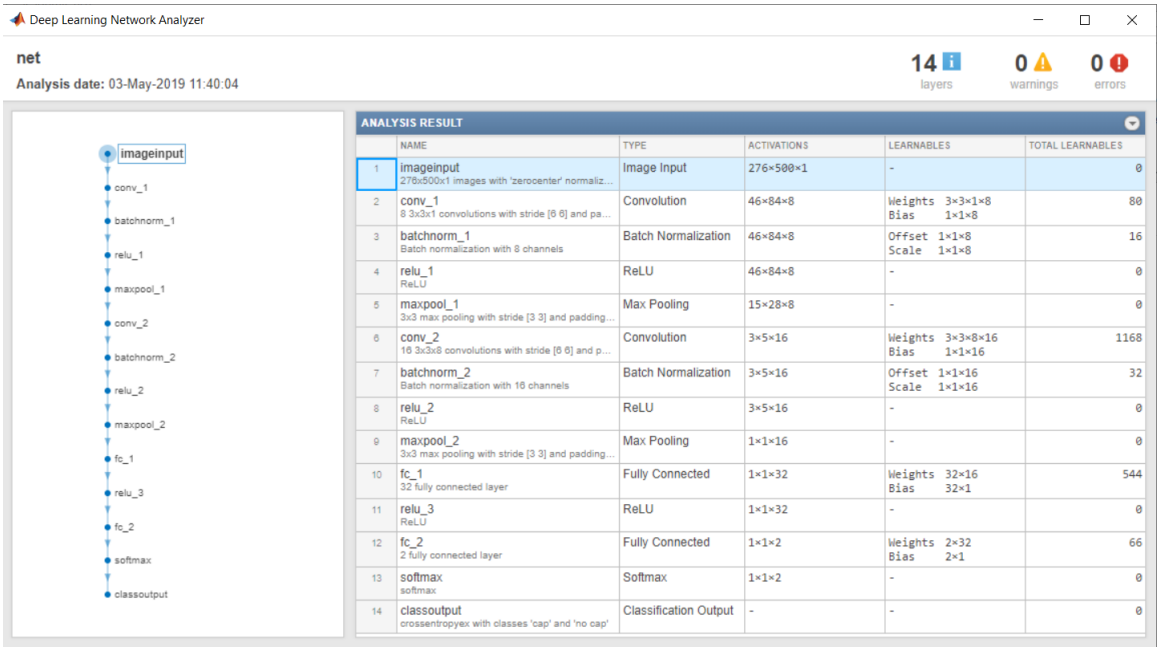


Figure 55: Architecture for *Conv_6*.

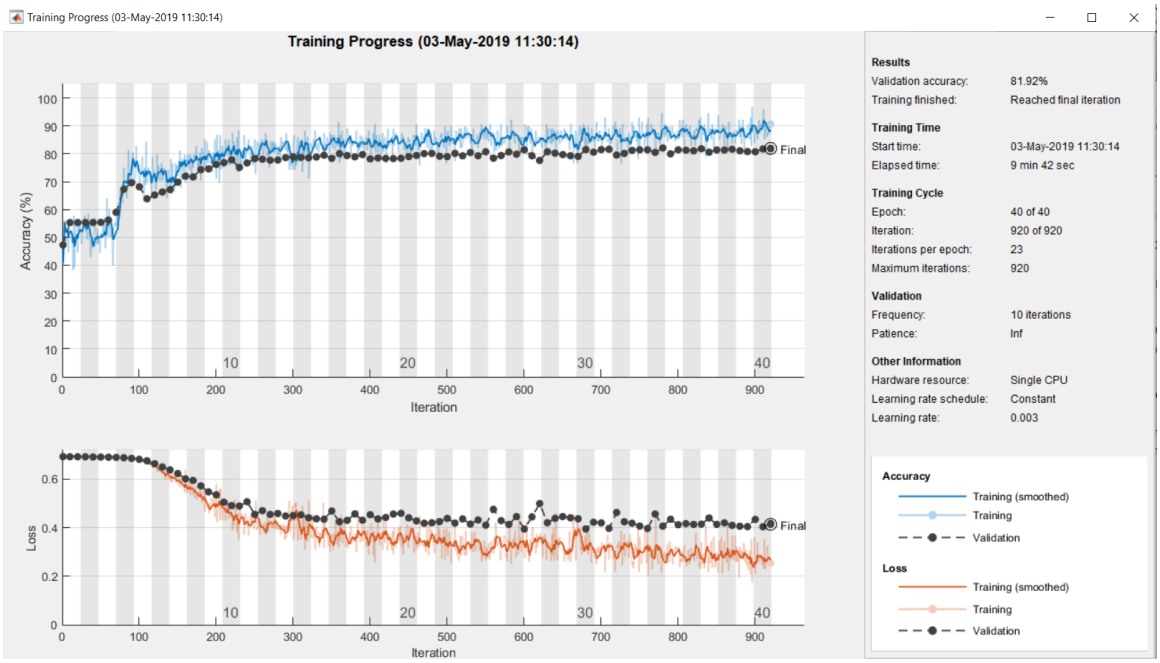


Figure 56: Training plot for *Conv_6*.

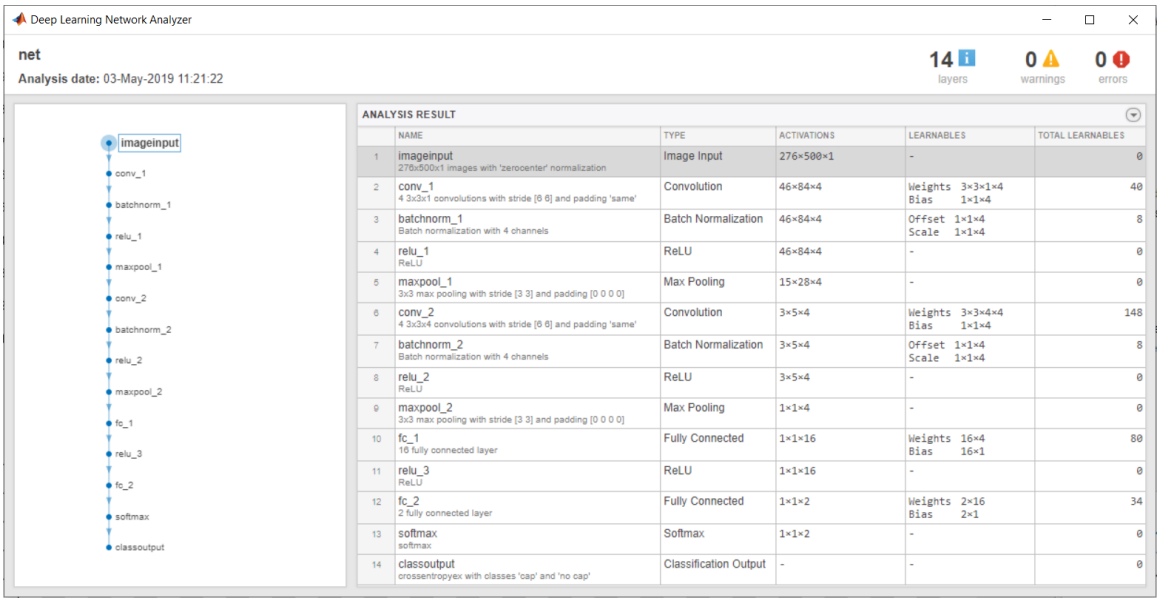


Figure 57: Architecture for *Conv_7*.

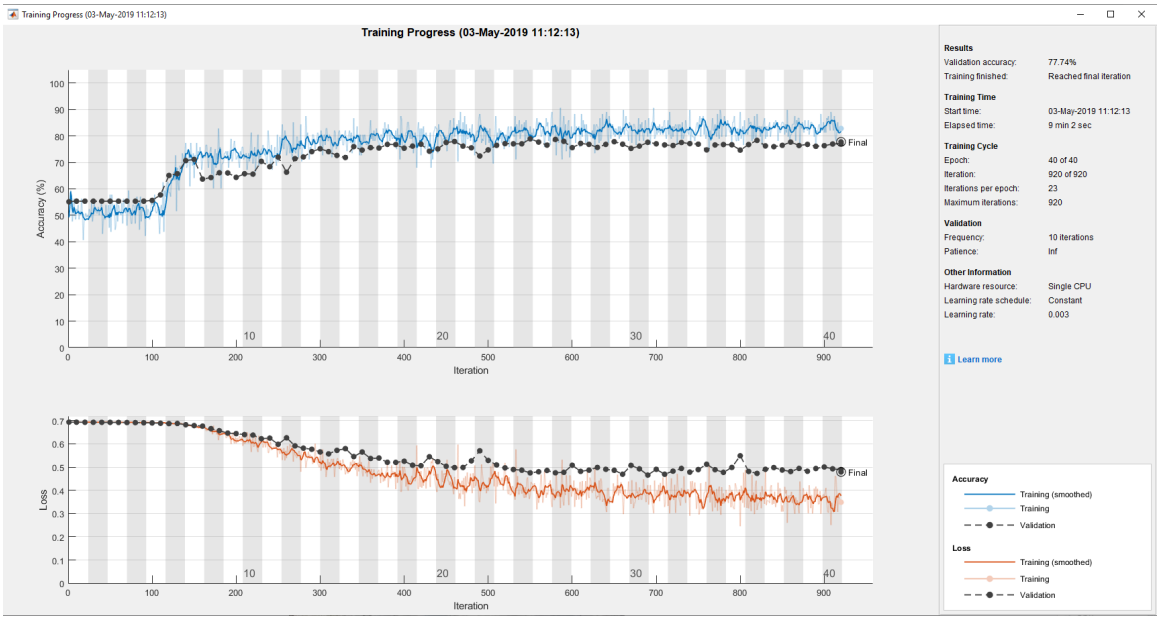


Figure 58: Training plot for *Conv_7*.

The architectures of the different networks are, as mentioned above, inspired by the AlexNet architecture. The receptive field can be increased by adding more convolutional layers, increase the stride and dilation and by adding pooling layers. To help increase the receptive field in the architectures, strided convolutions were applied. The number of convolutional layers and stride factor vary from architecture to architecture, but the objective is to get a large receptive field with few learnable parameters. All of the architectures have fully connected layers at the end of the model. In the fully connected layers every node is connected to the nodes in the previous layer. If the input size to the fully connected layers are big, the number of learnable parameters will be big. In the convolutional layers, the filter size determine how many nodes in the previous layer that are connected to a node in the current layer. The convolutional layers often have less learnable parameters than the fully connected layers. Therefore it was chosen to add strided convolutional layers and max pooling layers to reduce the spatial dimension before the fully connected layers to keep the number of learnable parameters low. As stated earlier, the number of learnable parameters is what decides the size of the network.
For this classification problem it is important to keep the size of the network small, but a small network is of little importance if the network performs poorly in terms of accuracy. The learnable parameters of the network are what the network learns during training, and it is important to not reduce the number of these parameters too much. As it can be seen in figure 44, the networks with the most learnable parameters are the networks that performs best in terms of accuracy. The classification time increased as the number of parameters increased, but it is important to mention that this is not always the case, which will be shown in section 4.3.3.

The network *Conv_1* is the CNN with the most learnable parameters. The architecture and training plot of this CNN can be seen in figure 45 and figure 46. The downsides of a larger number of learnable parameters are longer elapsed time during training and longer classification time, as well as a network of greater size which requires more memory. The upside of a larger number of learnable parameters is that the accuracy increases. The accuracy obtained in *Conv_1* (99.35%) is drastically higher than the smallest networks. It is important to have a high accuracy, but not at all costs.

Conv_2 obtained an accuracy of 98.49% with a lot less parameters. *Conv_2* is more than three times as small in size and classifies data samples at half the time as *Conv_1*. This makes *Conv_2* the most suitable CNN trained from scratch for the cap detection problem. *Conv_2* is fast and small, but still achieves a good accuracy.

As it can be seen in the training plots of the different CNNs trained from scratch in the figures 46,48, 50, 52, 54, 56 and 58 the validation accuracy is a bit lower than the training accuracy. Likewise, the validation loss is a bit higher than the training loss. The presence of this gap means that the networks performs better on the training data than on the validation data. This makes sense due to the fact that the data samples from the validation set are unseen data samples, while on the other hand, data samples from the training set were used during training. When the dataset is small, the training set will not represent every possible plastic bottle. Therefore it is good to see a small gap between the validation accuracy and training accuracy. If these accuracies were similar, suspicion would arise that the training and validation set were correlated. As stated earlier it is important to keep the training and validation set as separated as possible, therefore these datasets were manually collected with different plastic bottles. As the datasets grow bigger, and the training set represents a wider majority of plastic bottles, the gap between the validation and training accuracy would hopefully decrease.

4.3.3 CNN - Transfer learning

Transfer learning gave good results in terms of accuracy. The models deployed in this thesis was AlexNet, GoogLeNet and SqueezeNet. These models where trained on the ImageNet dataset and they vary drastically in size.
As explained earlier there are two approaches towards the transfer learning method. Either the classifier can use the exact same features as in the pre-trained model (freeze the weights) or the classifier can be fine tuned during training and make the features more specific towards the dataset at hand. In table 3 below, results of both approaches are shown and will be discussed. The architectures of each of the pretrained models will be presented as well as the training plots.

Transfer learning					
Name	Model and approach	Validation accuracy	Size	Learnable parameters	Classification time*
Transfer_1	AlexNet freeze weights	0.9971 99.71%	207 242 kB	61 million	0.0158 s
Transfer_2	SqueezeNet freeze weights	0.9899 98.99%	3 134 kB	1.2 million	0.0289 s
Transfer_3	SqueezeNet fine tuning	0.9993 99.93%	3 134 kB	1.2 million	0.0286 s
Transfer_4	GoogLeNet fine tuning	0.9902 99.02%	22 230 kB	7 million	0.0535 s

Table 3: Table of the results of CNNs trained with transfer learning. *Average time per data sample.

As shown in table3 all of the models and transfer learning approaches gave good results in terms of accuracy. The classification time may vary a bit from each time the model is run. These measured classification times can not be relied on when applying the networks on a different platform with different hardware. But since all of these times where measured on the same computer it gives a good indication of the classification time relative to each other. The networks in table 3 was run when using one single 8^{th} Gen intel I7-8650U CPU.
In the figures 59 to 61 the architectures of the pretrained models are shown.

ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES	TOTAL LE...
1	data 227x227x3 images with 'zerocenter' normalization	Image Input	227×227×3	-	0
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55×55×96	Weights 11×11×3×96 Bias 1×1×96	34944
3	relu1 ReLU	ReLU	55×55×96	-	0
4	norm1 cross channel normalization with 5 channels per element	Cross Channel Nor...	55×55×96	-	0
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27×27×96	-	0
6	conv2 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	27×27×256	Weights 5×5×48×256 Bias 1×1×256	307456
7	relu2 ReLU	ReLU	27×27×256	-	0
8	norm2 cross channel normalization with 5 channels per element	Cross Channel Nor...	27×27×256	-	0
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13×13×256	-	0
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×384	Weights 3×3×256×384 Bias 1×1×384	885120
11	relu3 ReLU	ReLU	13×13×384	-	0
12	conv4 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×384	Weights 3×3×192×384 Bias 1×1×384	663936
13	relu4 ReLU	ReLU	13×13×384	-	0
14	conv5 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×256	Weights 3×3×192×256 Bias 1×1×256	442624
15	relu5 ReLU	ReLU	13×13×256	-	0
16	pool5 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	6×6×256	-	0
17	fc6 4096 fully connected layer	Fully Connected	1×1×4096	Weights 4096×9216 Bias 4096×1	37752832
18	relu6 ReLU	ReLU	1×1×4096	-	0
19	drop6 50% dropout	Dropout	1×1×4096	-	0
20	fc7 4096 fully connected layer	Fully Connected	1×1×4096	Weights 4096×4096 Bias 4096×1	16781312
21	relu7 ReLU	ReLU	1×1×4096	-	0
22	drop7 50% dropout	Dropout	1×1×4096	-	0
23	fc8 1000 fully connected layer	Fully Connected	1×1×1000	Weights 1000×4096 Bias 1000×1	4097000
24	prob softmax	Softmax	1×1×1000	-	0
25	output crossentropyex with 'tench' and 999 other classes	Classification Output	-	-	0

Figure 59: Architecture of the AlexNet model.

Since AlexNet was announced the winner of the 2012 ILSVRC, the architecture of AlexNet have been a highly considered architecture. Alex Krizhevsky et al. showed that the high performance of the network was highly dependent on the depth of the model. In other words how many filters that are applied in each layer. This results in a network with many learnable parameters.

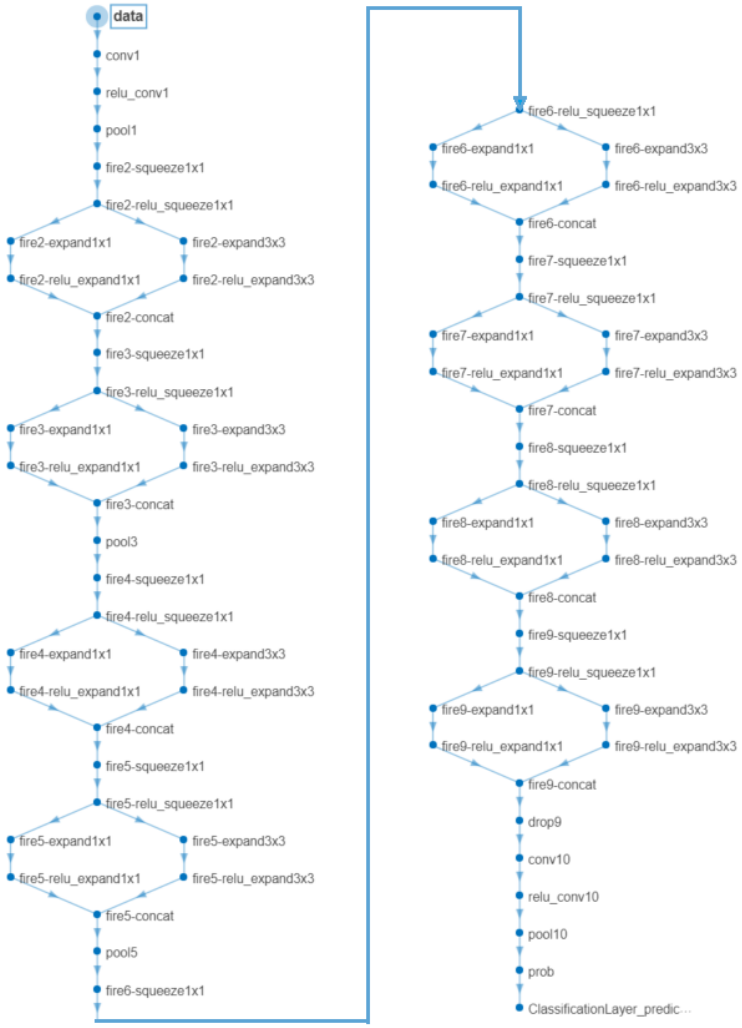


Figure 60: Architecture of the SqueezeNet model.

The main objective of the SqueezeNet model was to maintain a high accuracy network but with a lot less parameters. Forrest N.Iandonla et.al [17] created SqueezeNet and introduced what is called fire modules. The idea behind these fire modules is to reduce the number of parameters while maintaining high accuracy. The fire modules, as shown in the architecture of the SqueezeNet in figure 60, are the main building blocks of SqueezeNet. Since SqueezeNet have a lot fewer parameters than the other pretrained models that were applied for this thesis, it offers some advantages over the other models. There are at least three advantages for small networks like SqueezeNet. These advantages can be described in the following list [17].

- **Faster training**: Training time are proportional to the number of learnable parameters in the model. Due to the small (relatively speaking) amount of parameters in the model, the

network requires less time during training.

- **Less bandwidth with cloud updates**: It may be desirable to frequently upload an improved model to the RVM. Smaller networks requires less bandwidth when uploading models over the cloud.
- **Suitable for FPGA implementations**: FPGAs often have limited on-chip memory, and a small model like SqueezeNet may be implemented on FPGAs without the use of off-chip memory.

Figure 61: Architecture of the GoogLeNet model.

In figure 61 the architecture of GoogLeNet is shown. The first layers of the network are built up quite similar as the AlexNet architecture, but further out in the network there are added inception modules. These inception modules are what characterizes GoogLeNet. The idea behind these inception modules is to apply different kernel sizes at the same output. These kernel sizes are restricted to 1 x 1, 3 x 3 and 5 x 5, resulting in what can be interpreted as a wider network. These inception modules are stacked upon each other. The output from an inception module are depth concatenated [30].

In figure 62 to figure 65 the training plots from table 3 are shown. In the image description of these training plots, the same network names from table 3 are used. As explained earlier there are two approaches towards transfer learning. Either to freeze all weights and use the features originating from the original dataset used when training the pretrained model, or to fine tune these weights with the dataset at hand. Fine tuning a network often gives the highest accuracy, but there is a risk of overfitting. It also increases training time when fine tuning the weights. Freezing the weights and only performing feature extraction is recommended for very small datasets, e.g around 20 images from each class [25]. It can also be wise to apply a small learning rate so that the features found from the original dataset are not easily discarded.

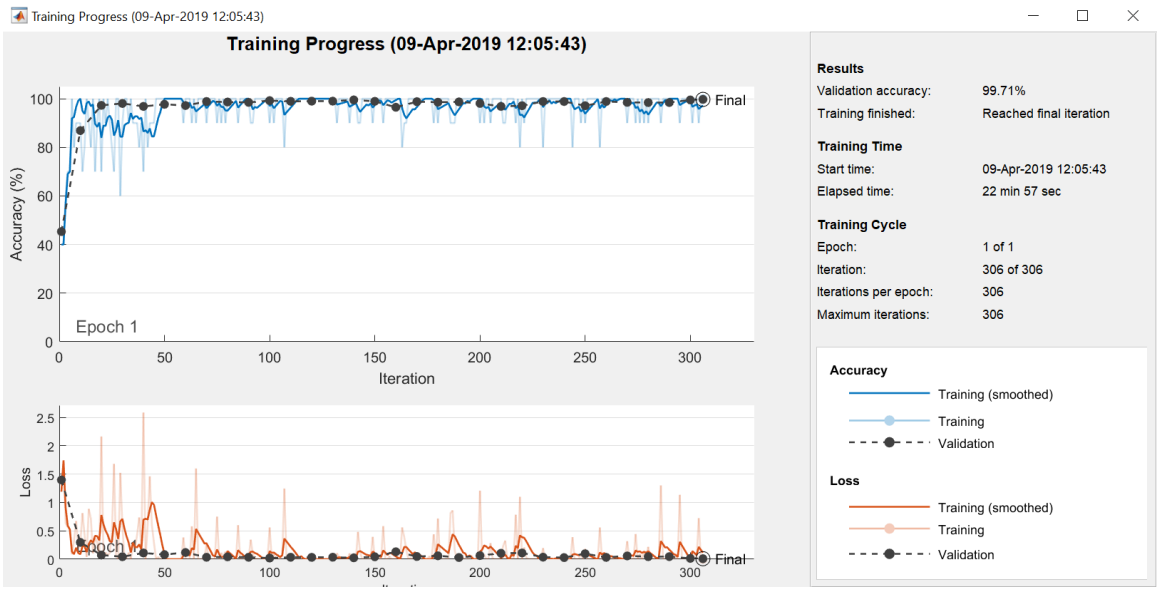


Figure 62: Training plot from the AlexNet model with all weights frozen. $Transfer\_1$ in table 3

The training plot of network $Transfer\_1$ from table 3 can be seen in figure 62. The AlexNet model was trained with frozen weights due to the depth of the network. The training time with fine tuning would take a lot of time, therefore it was chosen to freeze the weights of the network. The accuracy acquired was 99.71%, which are pretty descent considering the network have not seen any images from the dataset at hand during training. The optimization routine used was stochastic gradient descent with momentum (SGDM). The learning rate was kept constant during training and was set to 1e-4. The training routine lasted one epoch.

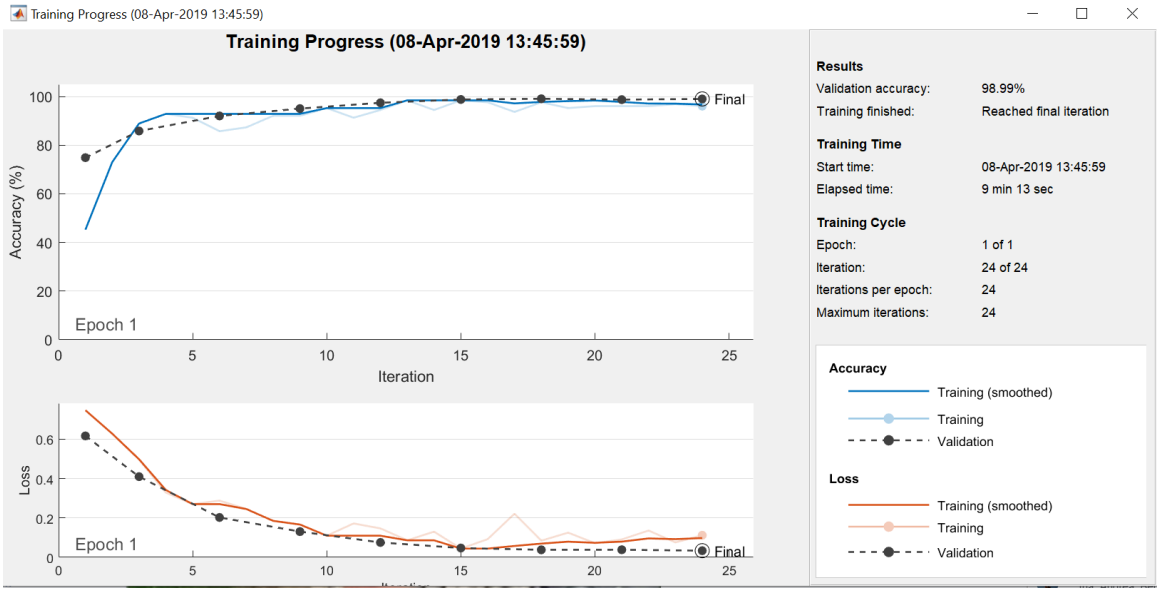


Figure 63: Training plot from the SqueezeNet model with all weights frozen. $Transfer\_2$ in table 3.

The training plot from network $Transfer\_2$ in table 3 can be seen in figure 63. The SqueezeNet model have just around 1.2 million parameters and it is manageable to perform both pure feature extraction with frozen weights and to fine-tune the model. In the training plot in figure 63 the weights are frozen. This classifier acquired a validation accuracy of 98.99%. The optimization routine used was SGDM. The learning rate was constant and set to 3e-4 and the training routine lasted one epoch.
As it can be seen in the plot, the validation accuracy is slightly higher than the training accuracy. In other words, the network predicts images that it has not seen before with a higher accuracy than images used to train the network. There can be several reasons why this phenomenon occurs. Two different reasons may be data augmentation on the training images or the use of dropout.

Data augmentation on training images: When training the network, image augmentation is applied on the training images. During training, the mini-batches of training data are augmented with a random translation of ± 10 pixels and with a 50% probability of reflecting the image horizontally. The validation images are not augmented, and this may be the reason for the validation accuracy to be higher than the training accuracy [3]. If the augmented training images have larger variations than the variations between the original training images and validation images it can be logical to think that the validation accuracy would be higher than the training accuracy. It is also worth mentioning that this network only had 24 training iterations. If the number of training iterations increase, the number of different augmentations on the training data would increase. This may result in a higher training accuracy, because the gap between the augmented training images would decrease. In practice this would mean that if the number of training iterations gets very high, the network has probably seen very similar augmentations at an earlier iteration.

Dropout: Dropout is a method used to address the problem of overfitting a network. This is done by randomly selecting certain nodes and their connections in the network and setting these to zero

for each training iteration. This makes the network co-adapt, and force it to not rely solely on a small selection of features [28]. During training, this means that some features are not activated and the network do not fully exploit its potential. The validation images runs through the full network. This may affect the phenomenon of a higher validation accuracy than training accuracy as the SqueezeNet architecture contains a dropout layer as seen in figure 60.

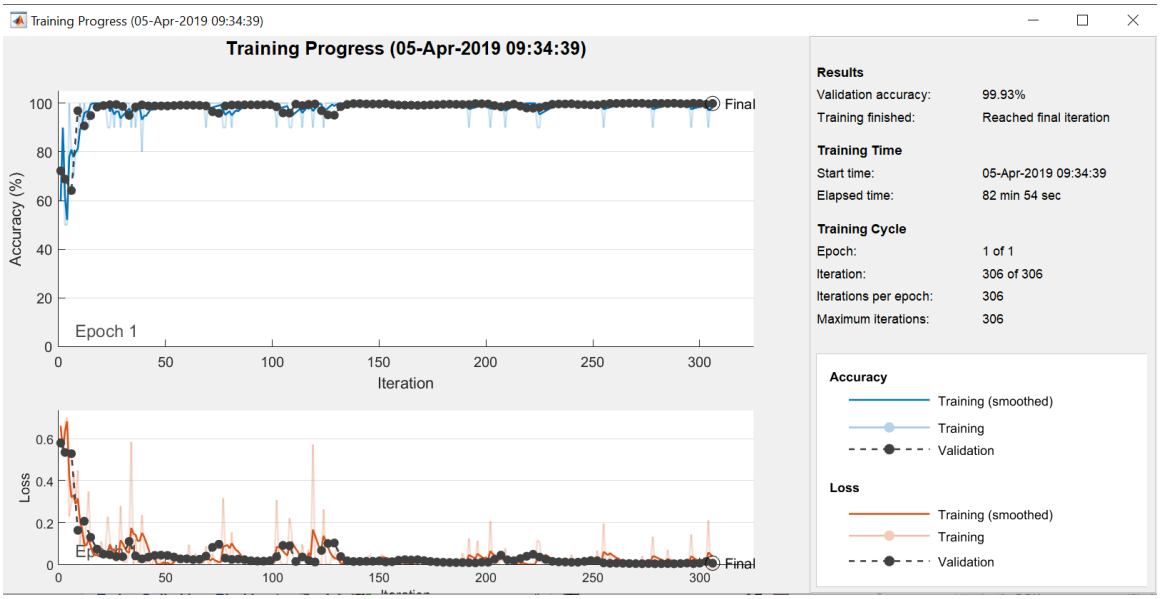


Figure 64: Training plot from the SqueezeNet model with fine-tuning. $Transfer\_3$ in table 3.

The training plot for network $Transfer\_3$ from table 3 can be seen in figure 64. This is the second classifier that uses the SqueezeNet model. The difference from the other SqueezeNet based classifier is that here the weights of the pre-trained model are fine tuned. The accuracy acquired is 99.93%. This is so far the best classifier in terms of accuracy. The optimization routine used was SGDM. The learning rate was constant and set to 3e-4. The training routine lasted one epoch.

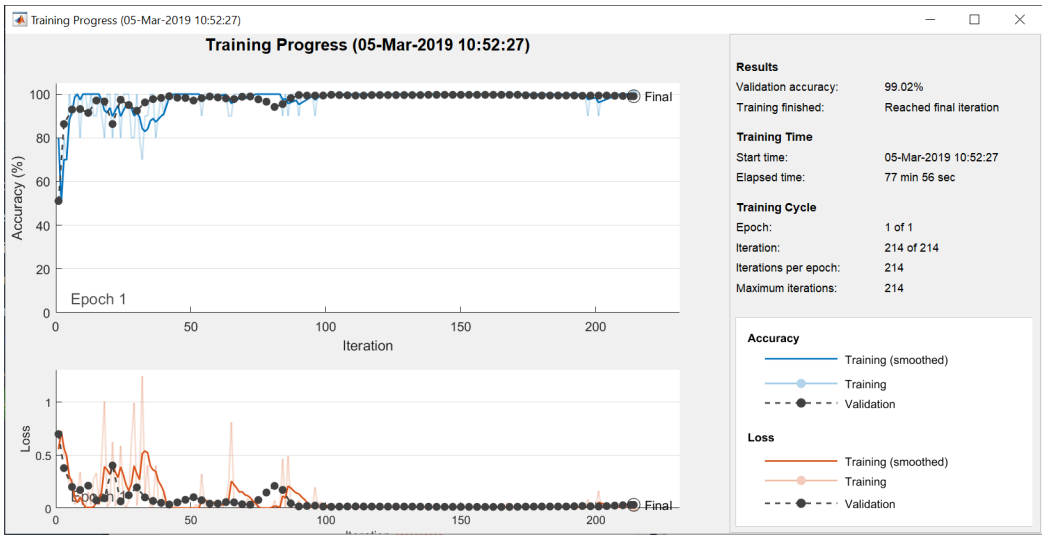


Figure 65: Training plot from the GoogLeNet model with fine-tuning. $Transfer\_4$ in table 3.

The training plot for network $Transfer\_4$ in table 3 can be seen in figure 65. This classifier fine tunes the weights in the GoogLeNet model and acquires a test accuracy of 99.02%. The optimization routine used was SGDM, with constant learning rate set to 3e-4. The training routine lasted one epoch.

All of the transfer learning networks performed very well in terms of accuracy. Fine tuning the weights proved to give better validation accuracy on the cost of longer training time. Since this is a real-time classification problem it is important to not only consider accuracy, but also to consider the size of the network and the time per classification.

AlexNet: AlexNet shows good results in terms of test-accuracy and time per classification, however it is the biggest model and it may be unrealistic to implement the network on a RVM.

SqueezeNet: SqueezeNet shows the best accuracy out of the three models that were applied. The model is small and can be implemented on most FPGAs, just using the on-chip memory. The time per classification are relevant for real-time classification.

GoogLeNet: GoogLeNet shows good test-accuracy and is relatively small in size. The time per classification is much longer than the other two models.

With the observations above in mind, the pretrained model that suits this classification problem best is SqueezeNet. The best result was achieved by fine tuning the weights. Although the training dataset is quite small, it only consists of two classes with around 1 500 images from each class. This is almost the same amount of images from each class as it were in the ImageNet dataset used at the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The networks do not seem overfitted, the training- and test-set are uncorrelated and consists of a variety of plastic bottles. With this in mind, the minimal gaps between the training and validation accuracy shows credible networks which are able to generalize and classify new samples with high accuracy.

Three important criteria is accuracy, size and classification time. SqueezeNet performs best of the three models in both accuracy and size, and performs second best in classification time. In table 3 the network that is best suited for the classification problem are network $Transfer\_3$. This network is based on SqueezeNet and fine tunes the weights with the plastic bottle dataset.

5 Conclusion

In this section the different results from the approaches applied during the thesis will be compared and discussed. Advantages and disadvantages of the different approaches will be made clear. A conclusion as to which method works best for the plastic bottle cap detection problem will be made.

From an early stage during this thesis it was quite clear that convolutional neural networks worked good, both when training from scratch and with transfer learning. The deep learning methods outperforms the more traditional pattern recognition methods when it comes to accuracy and classification time. That being said, there are a lot of pattern recognition methods that have not been tested, and it may exist undiscovered pattern recognition approaches that are suitable for the cap detection problem. It is time consuming to find and construct good pattern recognition methods. Due to the good performance of the deep learning approaches, the main priority during this study was to optimize the deep learning networks. On the basis of the methods implemented in this study, the deep learning approaches clearly performed best. One of the deep learning classifiers would be the best approach to implement on the reverse vending machine.

5.1 CNNs from scratch or with transfer learning?

What kind of deep learning learning approach that suits the problem best can be challenging to decide. CNNs from scratch gave best results in terms of classification time and size of the network, but the transfer learning approaches gave best results in terms of accuracy.

Embedded systems often have less computing power than a general purpose computer. This means that the classification time presented in table 2 and table 3 may be faster than what can be expected of an embedded system.
The accuracy for the different classifiers presented in this thesis may actually be better than the presented values. When calculating the accuracy of the classifiers during this thesis, it was not taken into consideration that there are several images which originates from the same plastic bottle. All of the images in the two datasets are considered completely independent. When a plastic bottle is inserted into a RVM, several images of the same plastic bottle are captured. When several images originating from the same plastic bottle are classified, the majority of the predicted classifications would be the chosen class affiliation. This means that the classification algorithm can predict the correct class for a plastic bottle even if there are some misclassifications as long as the majority of the classifications are correct.

With the statements above in mind, the best suited approach for the plastic bottle cap detection problem is the CNN *Conv_2*. This CNN was trained from scratch and is of size 1 882 kB. When implemented on an embedded system the classification time may be a bit longer than presented in table 2 due to different hardware, but the classification time is still quite short. When it is taken into consideration that several of the classified images originates from the same bottle, misclassifications might be overruled. If several images originating from the same bottle are classified, a majority vote would decide which class affiliation the plastic bottle have. This may result in a classification accuracy higher than what is presented for *Conv_2* in table 2.

It is also interesting to see that the accuracy obtained in *Transfer_3* (99.971%) is higher than the accuracy obtained in *Conv_1*, even though *Conv_1* has more learnable parameters and is of greater size. If the RVM can handle bigger networks and can cope with a slower classification time, transfer learning networks may be a better fit. This is due to the fact that *Transfer_3* have higher accuracy although it is a smaller network than *Conv_1*.

6 Further work

There are several ways to further develop the problem of detecting whether the cap is attached or not. In this section, these further developments will be mentioned.

6.1 Pattern recognition approach

There are several different image processing methods that were not tried out enough during this thesis and methods that were not tried out at all. In the subsections below there are suggestions to further develop the classification application.

6.1.1 Fast Fourier Transform

The idea behind deploying this method was the presence of periodic patterns. In images with the cap attached there are vertical lines on the cap that create a periodic pattern in the horizontal direction. Likewise, when the cap is detached there are horizontal lines on the plastic bottle that will create a periodic pattern in the vertical direction. This method was quickly tested during the thesis without very promising results. Due to time limitations the method had to give away to other methods. It is very possible that this method can obtain good accuracy due to the fact that the periodic patterns are almost present in every data sample in the dataset, i.e the features used by this method are very consistent.

6.1.2 Shape images

Shape images are the name of images where the whole bottle is captured from the side. In figure 66 such a shape image is visualized. These shape images were not included in the dataset. These images may be more consistent when it comes to location in the image and it may be easier to consequently locate the interesting part of the plastic bottle. The images in the dataset was captured by a ring of 6 cameras which yielded images from different angles at different time steps. The images in the dataset vary a lot when it comes to location in the image and rotation of the plastic bottle. This made it hard to locate points in the images that was general for all images included in the dataset. By using the shape images it may be easier to locate a point that is the same point on the plastic bottle for every data sample. For example, the tip of the plastic bottle can easily be found. In figure 66 the cap is detached and it can be seen that there are small taps on the tip of the plastic bottle. Due to the consistent location of the plastic bottle in the shape images, the location where these taps may be located can be found and used to differentiate between plastic bottles with the cap attached and detached.

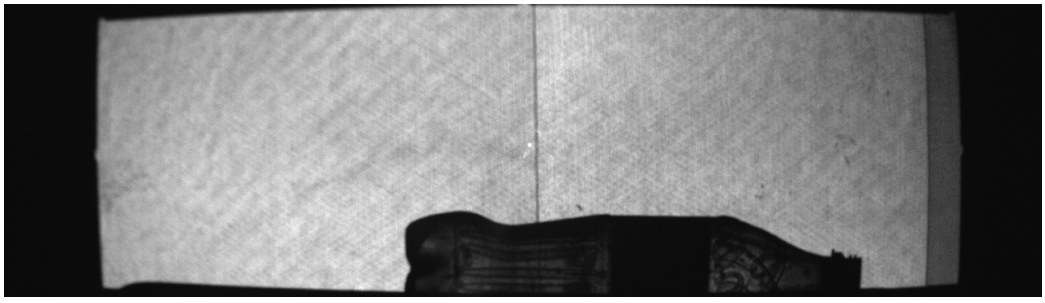


Figure 66: Shape image.

6.1.3 Prior knowledge

In this study the prior probabilities of caps and no-caps were 0.5. By personal experience, the caps are attached to the plastic bottles more often than they are detached when recycled. By observing recycled plastic bottles at different locations, prior knowledge of whether the cap is attached to the plastic bottle or not can be stated. This knowledge can be combined with the classifiers in this study to give better results in terms of classification accuracy.

6.2 Machine learning approaches

In this section, suggestions for further developing the machine learning approaches will be made.

6.2.1 Traditional machine learning

Very little time was spent on creating traditional machine learning algorithms. In light of the good results from the deep learning approaches, a decision to mainly focus on the deep learning classifiers were made. When creating traditional machine learning algorithms the features that are used in the dataset must be manually selected by a human expert. It can be tricky and very time consuming to find good features. The manually extracted features found during this thesis was not good enough and gave inferior results. For further work with traditional machine learning, good features needs to be found.

6.2.2 Deep learning

The transfer learning methods gave very good results. The best networks obtained accuracies close to 100%. These networks can not perform much better in terms of accuracy.
When training the networks from scratch there can be done some improvements. The dataset used during this thesis is, in the deep learning world, considered a small dataset. This dataset can be expanded to include many more images of plastic bottles. By making the dataset bigger the CNNs would most likely perform better without increasing the number of learnable parameters.
The architecture of the CNNs deployed in this thesis was made by hand. Different architectures can be tested to find more suitable architectures. It can both be time consuming and not very optimal to design architectures by hand. Automated neural architecture search can be deployed [14].

References

[1] URL: http://www.deeplearningbook.org/contents/optimization.html (visited on 05/23/2019).

[2] *Batch normalization layer - MATLAB - MathWorks Nordic*. URL: https://se.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.batchnormalizationlayer.html (visited on 04/29/2019).

[3] *Configure image data augmentation - MATLAB - MathWorks Nordic*. URL: https://se.mathworks.com/help/deeplearning/ref/imagedataaugmenter.html (visited on 05/09/2019).

[4] *Convolutional Neural Network*. en. URL: https://se.mathworks.com/solutions/deep-learning/convolutional-neural-network.html (visited on 04/29/2019).

[5] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/ (visited on 04/29/2019).

[6] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-2/ (visited on 04/29/2019).

[7] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/transfer-learning/ (visited on 04/29/2019).

[8] Rina Dechter. "Learning While Searching in Constraint-satisfaction-problems". In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. event-place: Philadelphia, Pennsylvania. AAAI Press, 1986, pp. 178–183. URL: http://dl.acm.org/citation.cfm?id=2887770.2887799 (visited on 04/29/2019).

[9] Meenal Dh and e. *What is the difference between AI , machine learning, and deep learning*. en-US. May 2017. URL: https://www.geospatialworld.net/blogs/difference-between-ai%ef%bb%bf-machine-learning-and-deep-learning/ (visited on 04/29/2019).

[10] *Dilated DenseNets for Relational Reasoning*. en. URL: https://www.groundai.com/project/dilated-densenets-for-relational-reasoning/ (visited on 04/29/2019).

[11] Chuong B Do. "The Multivariate Gaussian Distribution". en. In: (), p. 10.

[12] Idar Dyrdal. *Forelesninger - UNIK4690 - Vår 2016 - Universitetet i Oslo*. no. URL: https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger (visited on 04/29/2019).

[13] *Ellipse Detection Using 1D Hough Transform - File Exchange - MATLAB Central*. URL: https://se.mathworks.com/matlabcentral/fileexchange/33970 (visited on 05/13/2019).

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". In: *arXiv:1808.05377 [cs, stat]* (Aug. 2018). arXiv: 1808.05377. URL: http://arxiv.org/abs/1808.05377 (visited on 05/01/2019).

[15] *Fact Sheet: End Plastic Pollution*. en-US. Mar. 2018. URL: https://www.earthday.org/2018/03/07/fact-sheet-end-plastic-pollution/ (visited on 04/29/2019).

[16] *File:Colored neural network.svg*. en. URL: https://en.wikipedia.org/wiki/File:Colored_neural_network.svg (visited on 04/29/2019).

[17] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv:1602.07360 [cs]* (Feb. 2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360 (visited on 04/29/2019).

[18] *INF5860 – Maskinlæring for bildeanalyse - Universitetet i Oslo*. no. URL: https://www.uio.no/studier/emner/matnat/ifi/INF5860/index.html (visited on 04/29/2019).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 04/29/2019).

[20] Wenjie Luo et al. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 4898–4906. URL: http://papers.nips.cc/paper/6203-understanding-the-effective-receptive-field-in-deep-convolutional-neural-networks.pdf (visited on 04/29/2019).

[21] *Masaki Hayashi 2015, Autumn Visualization with 3D CG Digital 2D Image Basic. - ppt download*. URL: https://slideplayer.com/slide/10929748/ (visited on 04/29/2019).

[22] *Max-pooling / Pooling - Computer Science Wiki*. URL: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling (visited on 04/29/2019).

[23] *OpenCV: Canny Edge Detection*. URL: https://docs.opencv.org/3.4.3/da/d22/tutorial_py_canny.html (visited on 04/29/2019).

[24] *overfitting | Definition of overfitting in English by Oxford Dictionaries*. URL: https://en.oxforddictionaries.com/definition/overfitting (visited on 04/29/2019).

[25] *Pretrained Deep Neural Networks - MATLAB & Simulink - MathWorks Nordic*. URL: https://se.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html (visited on 04/29/2019).

[26] *Random forests - classification description*. URL: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#intro (visited on 05/15/2019).

[27] *Sobel Derivatives — OpenCV 2.4.13.7 documentation*. URL: https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html (visited on 04/29/2019).

[28] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[29] *Support Vector Machines for Binary Classification - MATLAB & Simulink - MathWorks Nordic*. URL: https://se.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html (visited on 05/14/2019).

[30] Christian Szegedy et al. "Going deeper with convolutions". en. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. URL: http://ieeexplore.ieee.org/document/7298594/ (visited on 04/29/2019).

[31] *The University of Auckland - COMPSCI 373 S1 C - Patrice's Lectures*. URL: https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/ (visited on 04/29/2019).

[32] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3320–3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf (visited on 04/29/2019).