University of
South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2025
Industrial IT and Automation

Anomaly Detection for Packaging Machines Using Machine Learning

Anders Lysgaard Lemme

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2025
Title: *Anomaly Detection for Packaging Machines Using Machine Learning*
Pages: *66*
Keywords: *Packaging machines, Computer vision, Image comparison, Convolutional Neural Network*

Student: *Anders Lysgaard Lemme*
Supervisor: *Ru Yan, Ola Marius Lysaker*
External partner: *Tronrud Engineering*

Summary:
Tronrud engineering's packaging machines have occurrences of lost products that exceed the expectations of their customers, and there is currently no optimal way to detect it. The goal of this thesis is to use computer vision to solve this problem by developing two different computer vision models. Where Model 1 will use image comparison to evaluate the similarity between an input image and a target image to detect product loss. Model 2 are using a convolutional neural network to extract features from an image. The features extracted in the network are then used to distinguish if there are any errors or anomalies in the product in the image. The implementation includes data collection and methods to preprocess the data such as grayscaling, thresholding, cropping, and rotating before the models use this data. With the data collected, both Model 1 and Model 2 were able to get 100% prediction accuracy on the output from the test data. However, for Model 2, the data collected was limited and the model should be trained and tested on a bigger dataset. Deployment of the models connected to live data from the machines was a shortcoming in this thesis, but suggested solutions for deployment and implementation are given for further work.

Preface

This master's thesis is carried out by an industrial IT and automation engineering student at University of South-Eastern Norway (USN). This study program is a collaboration with the industry where I have had part-time employment at Tronrud Engineering, which also is where the master's thesis has been carried out. Machine learning is a hot topic both in education and the industry, which made it interesting, both for Tronrud to investigate the possibilities to improve their production quality with the use of machine learning, and for me to learn more about the topic. Throughout the study program I have been introduced to machine learning and artificial neural network, with some basic tasks. This enticed me to learn more about the topic and develop a machine learning model to solve a real problem.

I would like to thank the supervisor Ru Yan along with co-supervisor Ola Marius Lysaker for their continuous assistance and the insight they have brought on the topic of this thesis. I would also like to thank Erik Hjertaas for the opportunity to choose this project, together with my colleagues for help and support.

Porsgrunn, 10th June 2025

Anders Lysgaard Lemme

Contents

List of Figures

List of Tables

Nomenclature

Symbol	**Explanation**
AI	Artificial Intelligence
ANN	Artificial Neural Network
CAD	Computer-aided design
CNN	Convolutional Neural Network
EM	Equipment module
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
KPI	Key Performance Indicators
MSE	Mean Squared Error
PLC	Programmable Logic Controller
PSNR	peak signal-to-noise ratio
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
ROI	Region Of Interest
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
TE	Tronrud Engineering
TCP	Tray and Case Packer
VPN	Virtual Private Network

1 Introduction

Experimentation with computer vision started in the 1960s and shortly after artificial intelligence(AI) quickly became an academic field of study, this was the beginning of an era using AI in computer vision. Today computer vision is a vast subject with many different objectives and solutions in areas like industry, transportation, healthcare, entertainment and many more. Typical tasks solved by computer vision are image classification, image segmentation, pose estimation, de-noising, Object detection- and tracking. The overall objective of this master's thesis is to use computer vision to improve the production quality of the machines produced at Tronrud Engineering packaging technology department. The objective is separated into two parts where each part includes development of a model with techniques used in computer vision.

- Model 1 Bag detection: develop an algorithm to compare images against a target image to detect differences that occur in undesirable situations.
- Model 2 Anomaly detection using CNN: develop a machine learning algorithm to classify if there are any errors or anomalies with the products inside an image.

Although Convolutional Neural Networks (CNN) was invented in the 1980's, it became a common method for image classification and other visual processing tasks after AlexNet, a CNN created by Alex Krizhevsky won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, with a significant improvement from the earlier years of the challenge [1]. Model 2 is a CNN-based algorithm to detect anomalies and errors using image classification and Model 1 is an image comparison algorithm to detect bags. These two models and their tasks are further described in Section 1.3 after some background information about the machine.

This master's thesis is a continuation of the master project: Anomaly Detection and Maintenance for Chips Packaging Machines using Machine Learning [2]. The master project analyzed the usage of sound and visual data to detect anomalies, the findings in image processing and computer vision will be the starting point for this project. These findings revolved around using preprocessing techniques to reduce the image size and removing features from raw image that shouldn't be characteristics distinguishing the classes.

1.1 Background

Tronrud Engineering's packaging department specialises in creating equipment and solutions for packaging, mainly in the food industry. Throughout the years, Tronrud has developed and produced a wide specter of packaging machines for different purposes. The third generation all-in-one tray and case packers (TCP) are produced the most of these machines. The TCPs are continuously improved to optimize production and quality. The TCP is classified as a factory automation machine controlled by a programmable logic controller (PLC). In addition, many of Tronrud's machines are equipped with an Internet of Things (IoT) solution that collects and stores data on a cloud-based platform for the purpose of analysis and monitoring. The data collected are general information about the machine state, Key Performance Indicators (KPI) and analytic data e.g. ambient temperature, air consumption, energy flow and servo torque values.

From the product infeed of the machine to the case/trey outfeed of the machine, the TCP has occurrences of product loss. The current product loss is larger than the acceptable average loss from the customers. Tronrud Engineering is interested in investigating and testing the possibilities of using computer vision to detect if there are missing products, as well as anomalies when filling cases/trays with products.

1.2 TCP Process Description

The TCP get products from an infeed, mainly bagged chips. The bags are fed into the machine, and packed into cardboard cases or trays. The machine is segmented into Equipment Modules (EM) that each handle one specific segment of the machine. Some of the EMs are simple, and only include small operations and sequences, while others are more complex controlling a bigger section of the machine. Figure 1.1 shows a process overview of the TCP. The blank magazine is the infeed for the cardboard blanks, and the product infeed is the infeed for the products. The product bags are pushed from the Pattern Builder into the folded case at the Empty Case Handover. The case outfeed is where the finally packed products are delivered.

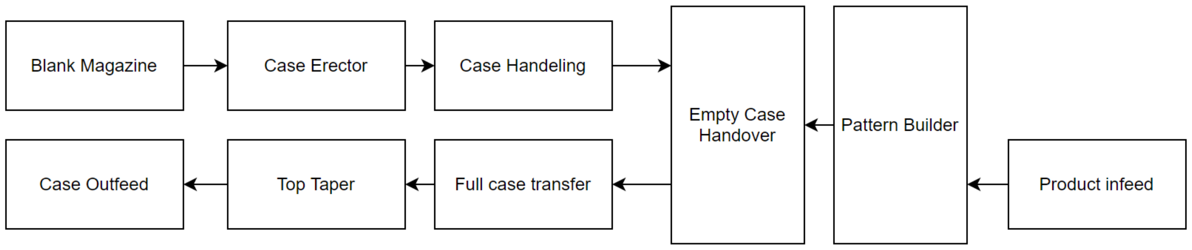


Figure 1.1: Process overview of the TCP, including all the equipment modules of the standard TCP and the product flow of the machine. The blanks are loaded in the Blank Magazine and the product infeed is where the products (bags) enter the machine.

The EMs shown in Figure 1.1 include logic for resetting sequence, homing of motors, settings for different products, machine state operations, and logic for alarms and warnings in addition to execute sequence. The EMs are described below, focusing on the execute sequence as it is the functionality of the process in the execution state.

- **Blank Magazine** is where the cardboard blanks are loaded. From the magazine, the blanks are moved with a conveyor belt to the Case Erector one by one.
- **Case Erector** is the equipment module that folds the blank into a box/case and tapes it in the bottom. The Case Erector has two ”arms” that are moved with servos in a vertical axis. The lower axis has plate with suction cups that uses vacuum to lift the blank upwards. This forms the blank into a cube shape. And then, the lower axis delivers the case to a side belt. From the deliver position of the lower axis, the sidebelts and the upper arm axis lift the case to the top of the machine, and the bottom of the case is folded and taped.
- **Empty Case Handover** is a relatively simple module that picks the case from the case erector and delivers it to Case Handling EM. The case is first picked up with a vacuum plate, and then the case is rotated 90° around the vertical axis and then is rotated 180° around the horizontal axis to the delivery position.
- **Case Handling** takes the case from the Empty Case Handover and places it in the filling position, after the case is filled with products it is delivered to the Full Case Transfer. This module also consists of two servo-driven axes, upper and lower. The upper axis picks the case in Empty Chase Handover's delivery position and delivers it to the lower axis in the filling position in front of the Pattern Builder. The Pattern builder has a pusher that pushes the products inside the case. Between each push the lower axis steps down to make room for a new layer inside the case until it's full. The filled case is then moved down to the Full Case Transfer EM.

- **Product Infeed** , feeds the products into the TCP with the help of conveyors. The execution sequence mainly controls the conveyors in relation to the product feed from the filler(s) in front of the TCP and the TPC's capability to receive products. The Product infeed also has different options for measuring the bags, which are chosen by Tronrud's customer's needs.
- **Pattern Builder** receives the products from the Product Infeed. The products are dropped down on a conveyor with three belts, called multibelt. The multibelt stacks up the products by moving one step for each bag that is dropped down on the belt. It creates the layers of products which is pushed inside the case. The belt with chips is then moved in front of the case which is ready for production. In this position, the machine has a servo-driven pusher that pushes the layer of products inside the case. The number of bags in each layer and the number of layers inside the case is recipe dependent and can be different for each product type.
- **Full Case Transfer** is the EM that transfers the case, after the case is filled at the Case Handling module the case is delivered to the Full Case Transfer. In this module, the case will be transported to the Top Taper with the help of roller conveyors and a servo-driven pusher. This module can also rotate the case so that the case are correctly placed to be sealed in the Top Taper.
- **Top Taper** uses cylinders to close the case flaps, and are then taped.
- **Case Outfeed** simply controls conveyors to move the cases to the outfeed of the TCP to the next machine in the production line.

1.3 Problem Description

As mentioned above, the problem is divided into two, where two different models will be created. The main task of these models are to detect product loss described in the Background Section 1.1.

Model 1: Bag detection. It will be located at the Pattern Builder, where bags are pushed inside the cardboard case. When the products are pushed inside the case, sometimes the product fails to be pushed inside correctly. To detect the products that aren't pushed correctly inside, the first model will take an image after each push. Each image will be compared with a target image of normal operating conditions. If the image has a bag in the foreground, the difference in pixel values will be higher than if there is no bag. The higher values will be separated with a threshold value that distinguishes if there is a bag(s) there or not.

Model 2: Anomaly detection using CNN, will use a neural network to detect missing bags in the filled case before they leave the TCP. For this model a camera will need to be located in the Full Case Transfer module, where it can collect data of the filled case. In addition to detecting missing bag(s) in the case, this model should also be tested to detect anomalies like miss-arrangement of bags in the case and other errors on the case which can cause problems when closing it.

Throughout the report, the terms ”Good” and ”Bad” will be used to describe the output of the models:

- For Model 1:
  - **”Bad”** indicates that a bag is detected.
  - **”Good”** indicates that no bag is detected.
- For Model 2:
  - **”Bad”** indicates all errors when packing bags inside the case, this includes missing bags, miss-arrangement and other errors.
  - **”Good”** would then classify all instances of normal operating conditions where there are no bags missing and the product arrangement is OK.

1.4 Report structure

Chapter 2 describes the theory related to this project. Chapter 3 is the implementation chapter where the method and implementation are described in detail. This includes data collection and the two models, from raw data to finally working algorithm. The achieved results for both Models are represented in Chapter 4 followed by discussion in Chapter 5. Chapter 6 concludes the project and gives suggestions for improvements and further work.

2 Theory

This chapter will give a brief description of the theory behind the methods that are used for this project. The topics presented in this chapter are described in chronological order of how they have been implemented in the Methodology chapter Chapter 3.

2.1 Real Time streaming protocol

Real Time Streaming Protocol (RTSP) is a standard protocol to transmit real-time video and audio streams. The RTSP is an application layer protocol that enables client-server communication over the internet [3]. RTSP is based on HTTP uses HTTP's concepts of request-response protocol, which makes it compatible with existing HTTP networks. The RTSP protocol comes with many different components and features as listed below [4].

- **Options:** a request that determines the request types that the server supports
- **Describe:** retrieves a description of the media object.
- **Setup:** Sets up how the bitstream are transmitted.
- **Redirect:** The redirect request redirects the client to connect to a different media server.
- **Play:** This starts the specified media session.
- **Pause:** This pauses the media until play is requested.
- **Record:** The record request is used to start recording of the media.
- **Teardown:** This request terminates the RTSP session.

Figure 2.1 shows how the client establishes a media session to the RTSP server, starts playing the media and terminates the session. In addition to the components described above the protocol uses Real-time Transport Protocol (RTP) which is a standardized protocol to transmit media data over the Internet [5].

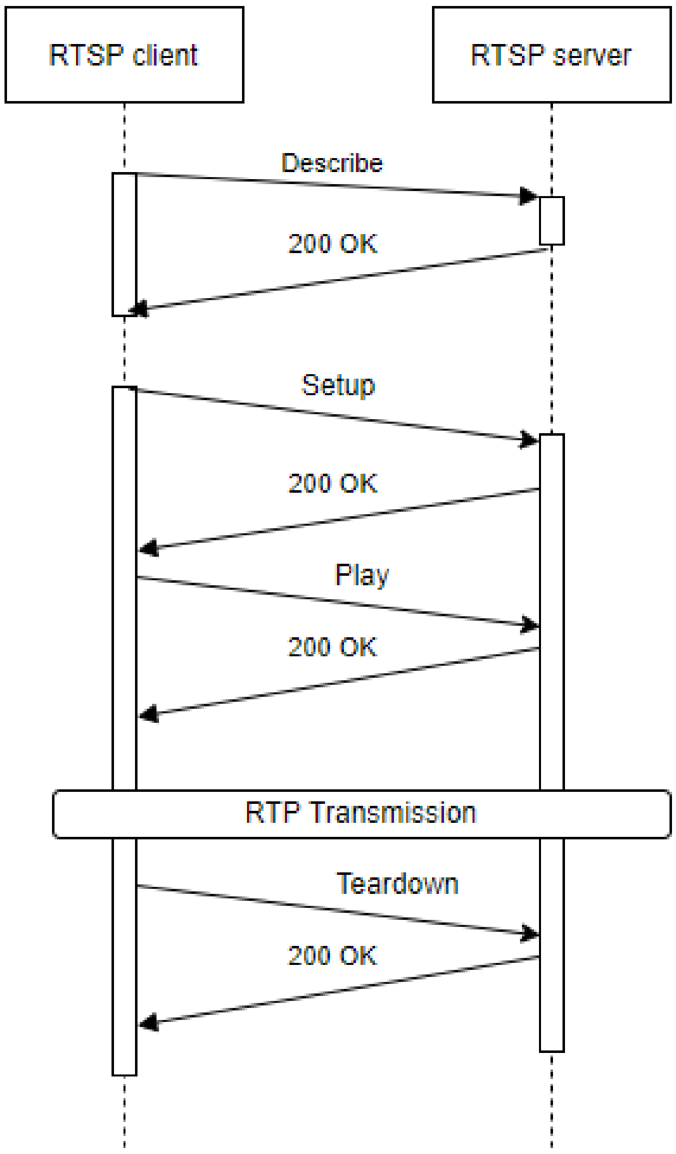


Figure 2.1: Establishment and media transmission with RTSP. (Adapted figure from [5])

2.2 Image Thresholding

Image thresholding is a form of image segmentation method that is typically used to separate foreground pixels from background pixels and is great in separating objects, which is useful in tasks like image recognition, image enhancement, and object detection. Image thresholding segments the pixel values in an image into two regions, where pixel values below a defined threshold are set to 0 and the values above are set to the maximum value, creating a binary image, expressed with Equation (2.1). This method is called binary thresholding but is also referred to as simple thresholding or two-level thresholding [6].

$$\text{if } f(x,y) \geq T \text{ then } f'(x,y) = max, \text{ else } f'(x,y) = 0 \tag{2.1}$$

where T is the threshold value separating the grayscale-level. $f(x,y)$ is gray-level pixel value of the input image at coordinate (x,y), and $f'(x,y)$ is the new pixel value ranging between 0 and max, where max is the maximum pixel value of the original image, typically 255 or 1 for normalized images.

2.2.1 Adaptive Thresholding

Due to varying lighting conditions, often binary thresholding isn't sufficient as it uses one global threshold value. Adaptive thresholding uses smaller regions of pixels in an image and computes the optimal threshold value for each region. Adaptive thresholding also computes the threshold values automatically. There exists a variety of methods for this. The grayscale histogram method is on of the commonly used. The grayscale histogram of an image is a histogram where the x-axis is represented with L grayscale intensity levels. The number of pixels at each gray level is denoted by n_k. In an ideal case for an adaptive thresholding method, there is a deep valley between two peaks in the grayscale histogram, where a threshold value $T(k) = k$ can be calculated having a value between 0 and $L-1$ separating the image into two classes. This threshold value is ideally at the bottom of the valley [6].

2.2.2 Otsu's Method

The Otsu thresholding method is also a global binary thresholding method. This method is named after Nobuyuki Otsu. Unlike normal binary thresholding, it automatically acquires an optimal thresholding value. The Otsu method computes the optimal threshold value $T(k) = k$ for $0 < k < L-1$ in a grayscale histogram. The threshold separates the grayscale intensity into two groups of pixels. This gives two classes, one with pixel values below the threshold k and another with pixel values above the threshold. Otsu's method uses the discriminant criterion measure, the measure that separates the classes to evaluate the separability of the threshold [7][6]:

$$\eta = \sigma_B^2/\sigma_T^2 \tag{2.2}$$

where σ_B^2 is the between-class variance and σ_T^2 is the total variance of the image. From Equation (2.2) we have an optimization problem that gives an optimal k when η is maximized. Since σ_T^2 is independent of k we get optimal threshold k^* for:

$$\sigma_B^2(k^*) = \max_{0<k<L-1} \sigma_B^2(k). \tag{2.3}$$

The between-class variance σ_B^2 can be calculated with the following formula:

$$\sigma_B^2(k) = \frac{(\mu_T\omega(k) - \mu(k))^2}{\omega(k)(1-\omega(k))} \tag{2.4}$$

where $\omega(k)$ is the zeroth order cumulative moments of the histograms probability distribution, $\mu(k)$ is the first-order cumulative moments of the histogram and μ_T is the total mean of the image [7].

2.3 Convolutional Neural Network

As mentioned in Chapter 1, the Convolutional Neural Network (CNN) is considered the first choice for image classification and it will also be the choice for Model 2, where an anomaly detection model is going to be developed with an image classification task to distinguish between ”Good” and ”Bad”. CNN's are effective in image classification because they reduces the number of parameters and extracting the features of the image with convolutional layers. CNN's share some common basic attributes with other Artificial Neural Network (ANN). The CNN is organized as a feed forward neural network where the output from the neurons in each layer is fed forward to the next layer for further processing. The CNN also uses back-propagation in training to update the weights. The output layer of the network has a loss function, which computes the margin of error between the output and the desired output then the weight is updated using the gradient of the loss from the previous layer back-propagated [8].

2.3.1 CNN architecture

The CNN architecture can have many variations, but they all have three types of layers as building blocks, convolutional layers, pooling layers and fully connected layers. Typically a stack of convolutional layers and a pooling layer followed by a fully connected layer makes up a CNN architecture. Figure 2.2 illustrates an example of a CNN architecture, where feature extraction or feature learning is done with convolutional layers and pooling layers, and the output is separated into classes with a fully connected layer.

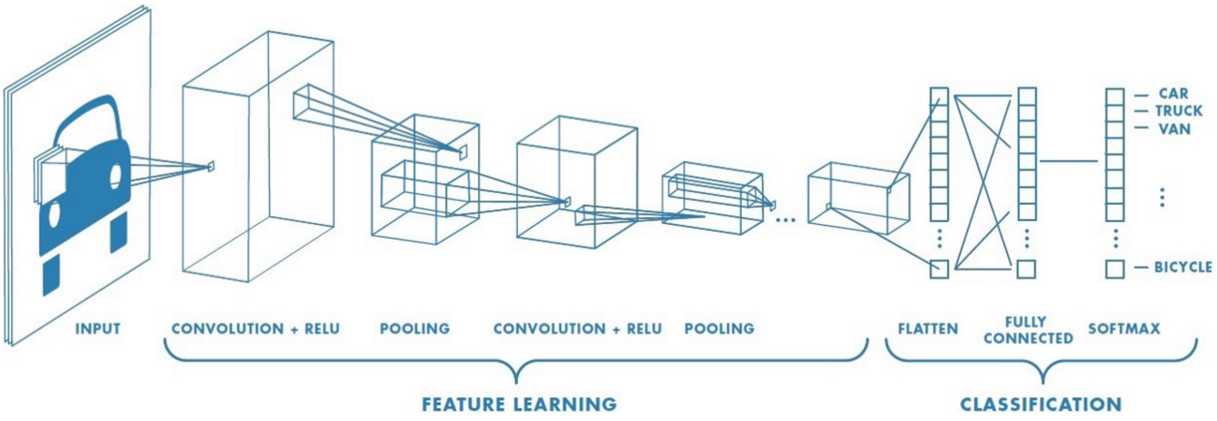
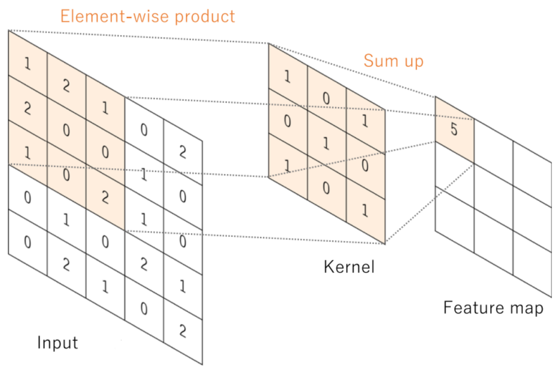


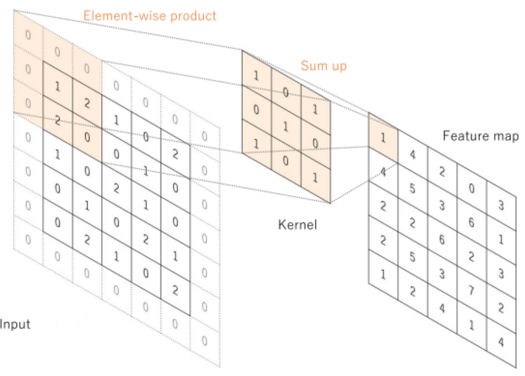
Figure 2.2: CNN architecture example from input image to output class. Here features are learned with combinations of convolutional layers followed by pooling layers, the final feature map is flattened and the features are then mapped with fully connected layers. The classification of the CNN is performed similarly to a standard artificial neural network[9].

Convolutional Layer

As the name implies the convolutional layer is a fundamental part of the CNN architecture. This layer use filters, often called kernels for feature extraction. The Kernel are small sized matrix of numbers that moves over the image as a sliding window shown in Figure 2.3a. Each step of the sliding window function are called strides, for each stride an output pixel is calculated by the dot product which is the sum of the product between the input and the kernel. This procedure is repeated with multiple kernels which creates a number of feature maps containing the extracted features of the input. The convolutional operation does not allow an overlap with the center of the kernels with the outermost elements of the input, this leads to reduced height and width of the feature map compared to the input. By introducing padding, typically zero-padding allows the center of the kernels to overlay with the outer elements of the input by adding rows and columns of zeros around the input. This makes the dimensions of the output feature map equal to the image dimension, which can be seen in Figure 2.3b.



(a) Element-wise product calculation between kernel and input image that extracts a feature map.



(b) Element-wise product calculation between kernel and input image with zero-padding.

Figure 2.3: (a) illustrates how the convolutional layer extracts features of the input image into a feature map. (b) shows how zero-padding on a 5x5 input image gives a feature map of the same size as output [10]

The outputs of the convolutions are passed through a nonlinear activation function to make the network nonlinear, this allows for more complex models. In the past sigmoid and hyperbolic tangent function was often used because of their biological neuron behaviour, but the most common nonlinear function used today is the rectified linear unit (ReLU), which simply is computed by Equation (2.5) [10].

$$f(x) = max(0,x) \tag{2.5}$$

Pooling layer

The pooling layers is an operation used for downsampling the feature maps and reduces the number of learnable parameters, which reduces the complexity of the model. The pooling layer works similarly to the convolutional layer, but without weights/learnable parameters. There are two main types of pooling operation methods used in CNNs, max pooling and global average pooling:

- **Max pooling**is the most common pooling operation which extracts patches of the feature map and outputs the maximum value, the remaining patches of the feature map are discarded. Typically this is done with kernel dimension of 2x2 and a stride of 2, this reduces the feature map by a factor of 2 making the number of pixel values in each feature map 25% of the input [10].
- **Global average pooling** is another pooling operation that has a different purpose. This pooling operation drastically downsaples a feature map down to a 1x1 array by taking the average of all the elements in each feature maps. This operation is

only used as a last step before the fully connected layer and allows variation in the input dimensions [10].

Fully connected layers

The fully connected layer also often called dense layer, has the objective to map the features extracted from the previous layers into output classes. Before the final feature map is passed to the fully connected layer(s) it is typically flattened into a 1D array in which leaning weights is used to connect all the inputs to every output. The fully connected layers are usually followed by the nonlinear activation function ReLU, except the last fully connected layer that has to be chosen based on the task. Typically sigmoid activation function is used for binary classification as it returns a value between 0 and 1 or -1 and 1. For multi-classification tasks, the softmax activation function is normally used, because this function normalizes the output into probability distribution.

2.3.2 TensorFlow and Keras

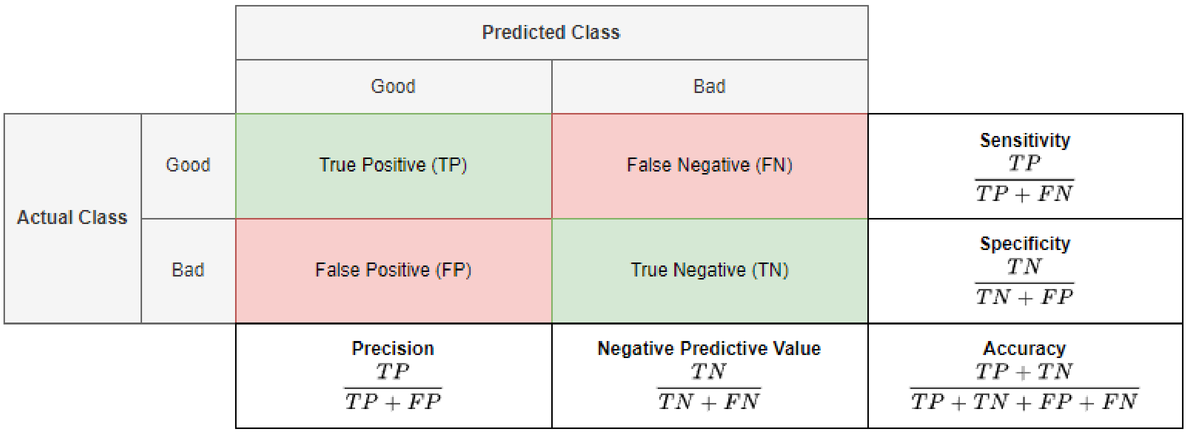
To develop the CNN model, Keras and TensorFlow has been used. TensorFlow is an open-source deep learning framework developed by Google, TensorFlow offers functionality for building and training models. TensorFlow has adopted Keras which makes Keras run on top of TensorFlow. Keras is a neural network library that provides APIs for building and training neural networks supported in Python.

2.4 Confusion Matrix

Since both Model 1 and Model 2 can be examined as a classification task, confusion matrix has been used to validate the performance of the algorithms. This matrix indicates in what area the algorithms is confused when making an output prediction. This is a great tool that gives insight to what types of errors that the algorithms make. The confusion matrix can seen in Figure 2.4. From the test data, this matrix is filled in with the correct predictions, true positive and true negative and the incorrect predictions, false positive and false negative. From the results of Model 1 and 2, ”Good” are the positive classifications and ”Bad” are the negative classifications. From this matrix, there are a number of rates that can be calculated to get insight in the performance of the models. The rates used to evaluate model 1 and 2 are listed below [11]:

- **Precision:** also called Positive predictive value, indicate how many times the prediction is correct, when the model prediction is ”Good”.

- **Negative predictive value:** indicate how many times the prediction is correct when the model predicts ”Bad”.
- **Sensitivity:** also called true positive rate, indicate how often the prediction is ”Good”, when the actual class is ”Good”.
- **Specificity:** also called false negative rate, indicated how often the prediction is ”Bad”, when the actual class is ”Bad”.
- **Accuracy:** is the overall calculation of how often the classification is correct.



		Predicted Class		
		Good	Bad	
Actual Class	Good	True Positive (TP)	False Negative (FN)	Sensitivity $\frac{TP}{TP+FN}$
	Bad	False Positive (FP)	True Negative (TN)	Specificity $\frac{TN}{TN+FP}$
		Precision $\frac{TP}{TP+FP}$	Negative Predictive Value $\frac{TN}{TN+FN}$	Accuracy $\frac{TP+TN}{TP+TN+FP+FN}$

Figure 2.4: Confusion matrix, used to evaluate the output of the algorithms [12].

3 Implementation

This chapter covers the methods and materials used to implement the models. Firstly, the data collection methods will be covered, including camera positions and connection, followed by the image comparison algorithm and at last, the anomaly detection with machine learning.

3.1 Data collection

To collect data a camera system that is already installed on many of the machines have been used. The camera system is called provision-ISR which is an IP camera system that transmits video footage through IP network. The camera system has up until now only been used for monitoring and maintenance of the TCP remotely. The camera position for the image comparison algorithm has been placed in the Pattern Builder where the pusher pushes products into the case. Figure 3.1 shows a figure of the camera setup, here the yellow dot represents the camera position and the dotted square represents the region of interest (ROI).

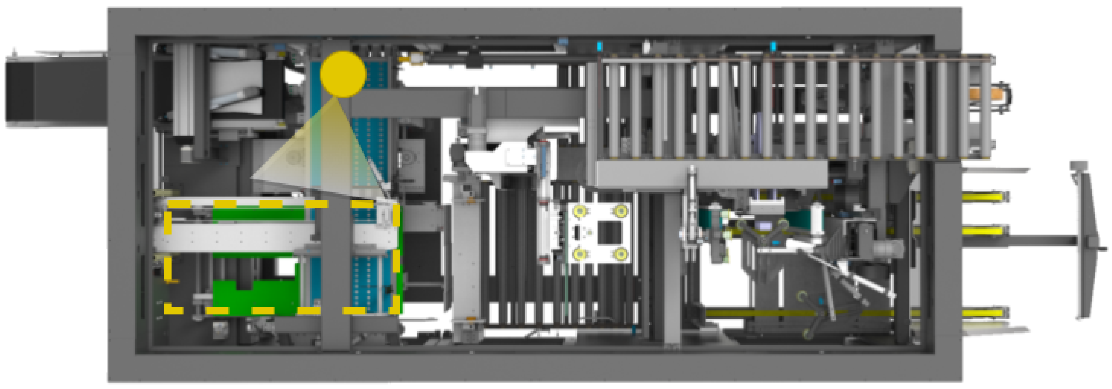


Figure 3.1: Camera 1 position in the TCP to collect data for image comparison algorithm. This 3D model is created in a Computer Aided Design (CAD) program and shows the TCP from a top view. The footprint of the TCP is $6.596m^2$.

The camera position for the anomaly detection algorithm is placed in the "roof" in the end of the full case transfer. This was the only position where nothing is covering up the sight of the camera and it's possible to see the full case to detect if there are any bags missing or other anomalies.

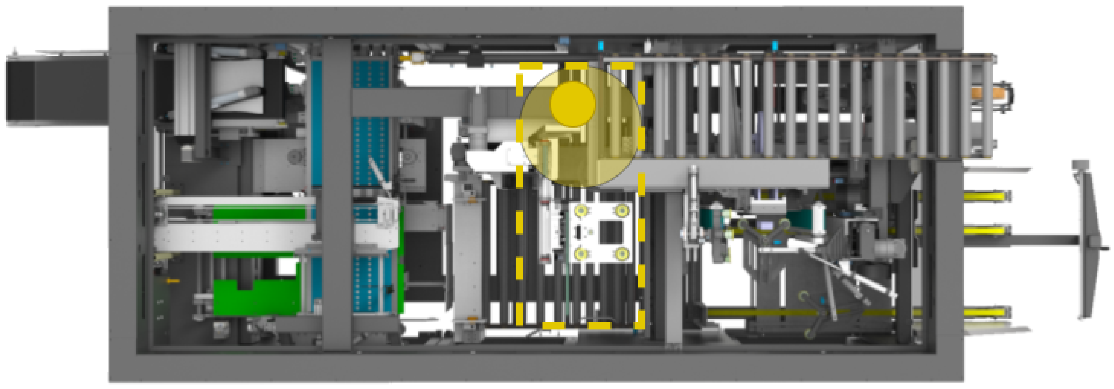


Figure 3.2: Camera 2 position in the TCP to collect data for CNN. This 3D model is created in CAD and shows the TCP from a top view. The footprint of the TCP is $6.596m^2$.

The camera system is hosted on the local network and it is necessary to connect to the local network to connect to the cameras. In the development period a VPN connection to the TCP has been used to collect data, but a computer should be connected directly to the system for a finished algorithm. Since there is no optimal solution to get the data directly from the Provision platform a script was created to connect to the camera with RTSP and record videos.

The script was created using OpenCV and os library, os is used to split the filename to determine what video format the data was saved. OpenCV has many built-in methods for computer vision, video and image processing. The two main functions used to capture and store the data are VideoCapture and VideoWriter. VideoCapture is used to create a video capture object and VideoWriter is used to create the video storing object. The input used to the VideoCapture method is the RTSP connection string to the IP cameras which is formatted *"rtsp:/[username]:[password]@IP:port/[channel ID]&[stream type]"*, username and password are the username and password to Provision camera system, the IP is the IP address where the system is hosted, and port 554 uses Transmission Control Protocol to establish connection and transmit data packets. In addition, the connection string has Channel ID and streaming type. VideoWriter's inputs are filename, video format, fps and dimensions. A installation of video codec might be needed for encoding and compression of the video format for VideoWriter to work.

3.2 Model 1: Bag detection

Before testing methods to develop this model some basic preprocessing was done to extract the ROI. Firstly, the raw images from camera 1, positioned in the pattern builder, are loaded to the program. The raw images include the area of the machine which is not interesting for the detection. To get the ROI the image was first rotated so that the ROI could simply be cropped out in a rectangular shape. The rotation was tested with a simple rotation function, some part of the ROI then was lost in the translation because it was located outside the dimensions of the original image size, which can be seen in Figure 3.3a. The function was extended to calculate the new bounding dimension and take translation into account, shown in Figure 3.3b.

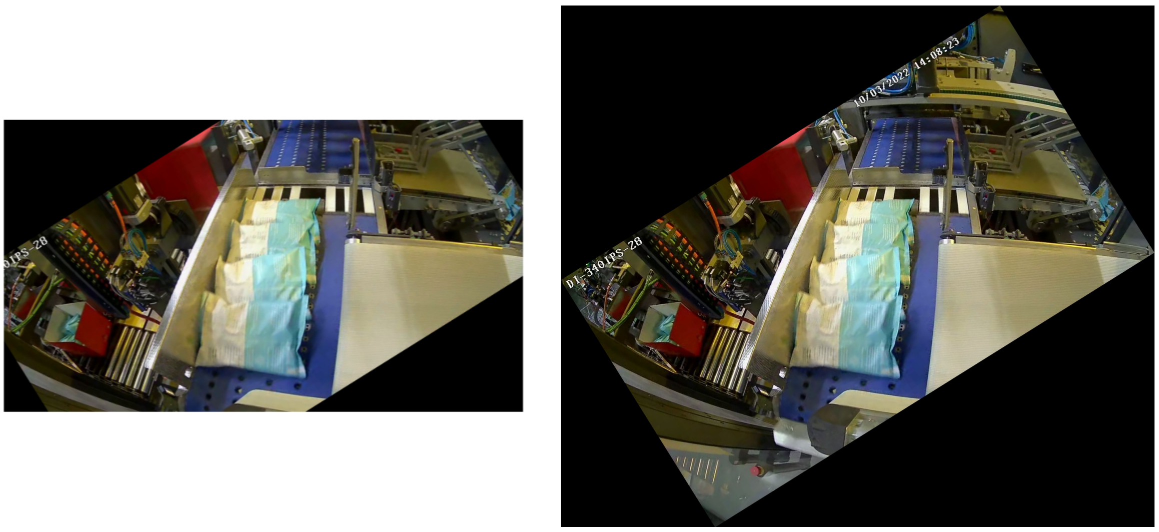
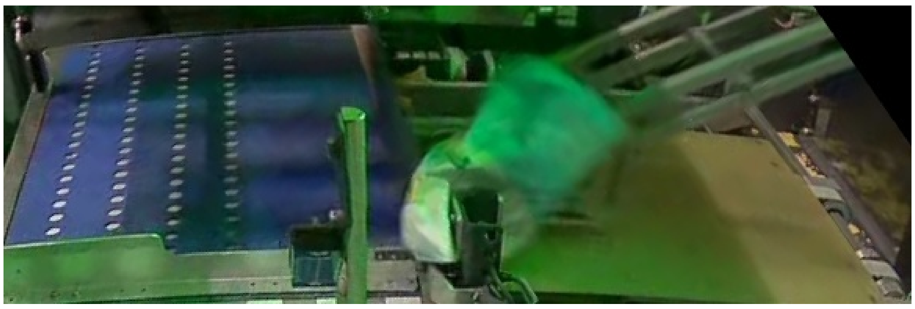

(a) Simple rotation. (b) Rotation with new bounds.

Figure 3.3: Example images that shows the difference between simple rotation with original image size (a), and the image with a new calculated bounding dimension (b).

The rotated images are then cropped. The ROI stretches from the opening of the case where the products are pushed inside and all the way back to the frame of the machine, behind the pusher. This area is the ROI because it is the region the bag will appear if it is not correctly pushed inside the case. The final chosen ROI of the image is rotated 35° counterclockwise and is a rectangular shape with pixels 270 to 460 on the y-axis and 720 to 1300 on the x-axis which makes the new image 190x580, as the examples shown in Figure 3.4.


(a) ROI example of erroneous image.

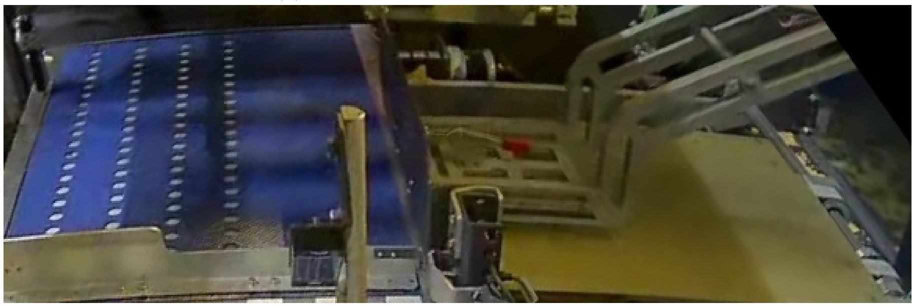

(b) ROI example of image without error.

Figure 3.4: Two images of the ROI of Model 1. (a) is an erroneous or ”Bad” example where a bag is in the ROI. (b) is a ”Good” example without any bags inside the ROI.

There are many different methods to compare images, and in the beginning of this project some of them was tested. One method is to compare the histograms, but as the pixel intensity changes between the ”Good” and the ”Bad” image was so small, this method was quickly eliminated.

Further testing was done with mathematical pixel calculations to try and distinguish the target image from erroneous images. With these test, mean squared error(MSE), root mean squared error (RMSE) and peak signal-to-noise ratio(PSNR) was tested on grayscale images shown in Table 3.1. The preferred outcome of these experiments would be that 1 and 2 had lower error than 3 and 4, since experiment 1 and 2 compare images without errors or bags, while 3 and 4 are between a OK and erroneous image. This is not the case, this is because there are variations in the images like shadows, color from the light, slightly offset in positions, blur and so on.

Table 3.1: Initial experiments with MSE, RMSE & PSNR of grayscale images

Experiments	MSE	RMSE	PSNR
1: Good 1 vs Good 2	98.28	9.91	28.20
2: Good 1 vs Good 3	378.46	19.45	22.35
3: Good 1 vs Bad 1	312.19	17.67	23.19
4: Good 1 vs Bad 2	508.14	22.54	21.07

3.2.1 Thresholding

To solve the problem described above, thresholding was applied to try and separate the bag from the background, and minimize variations like lighting conditions. Initially it was tested with global binary thresholding, adaptive mean thresholding and adaptive Gaussian thresholding which can be seen in Figure 3.5 and Figure 3.6, without and with a bag in the ROI, respectively.

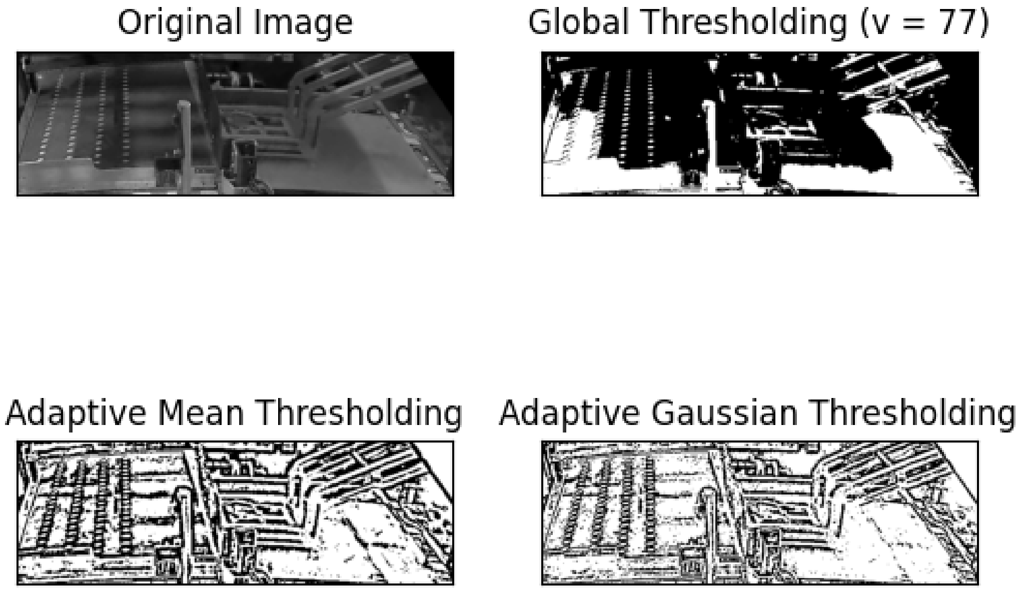

Original Image
Global Thresholding (v = 77)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Figure 3.5: Thresholding methods comparison with a ”Good” image

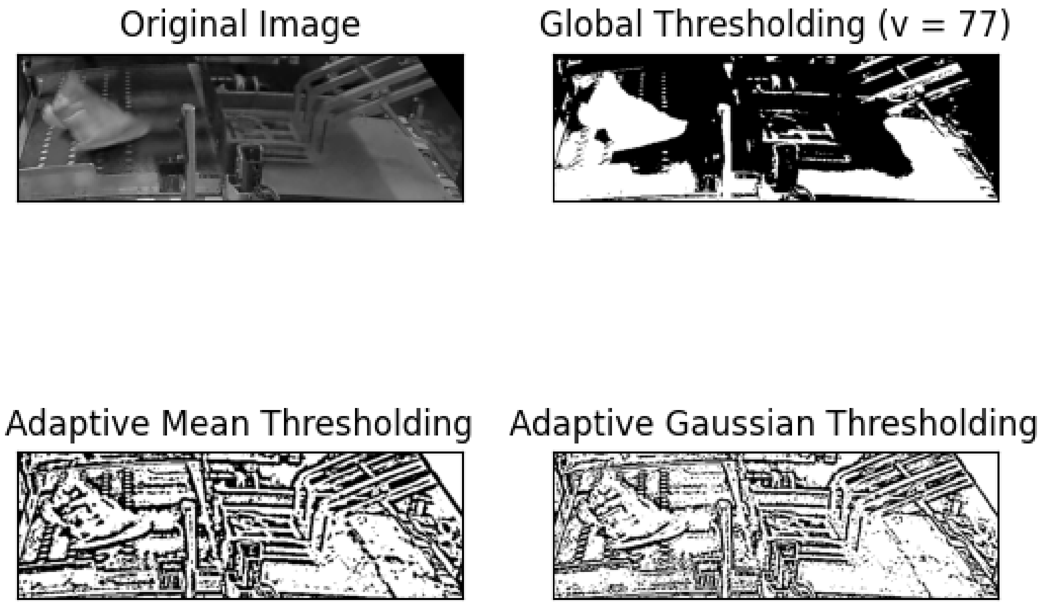
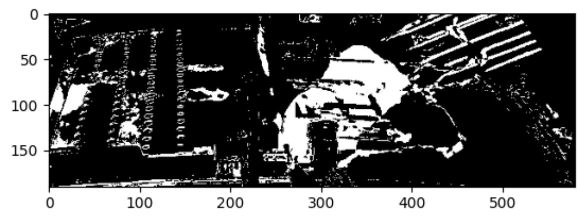

Original Image
Global Thresholding (v = 77)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Figure 3.6: Thresholding methods comparison with a "Bad" image.

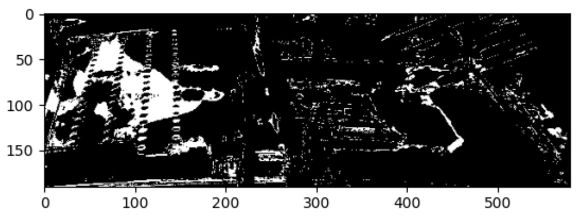
From these figures, the bag is best segmented with the binary global threshold, which gives a black background while the foreground is white. The other methods are better suited for images that has a cleaner background or for detecting edges. To automatically obtain an optimal global threshold value for all images, the simple binary thresholding method was replaced with Otsu's thresholding method.

3.2.2 Comparison

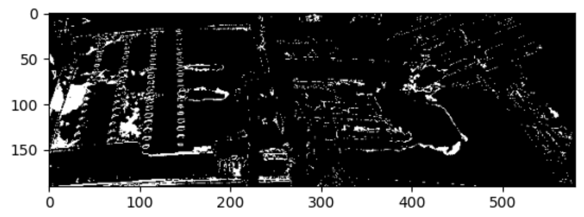
The absolute value of two images subtracted has been used to compare the segmented images with each other. This gives the value 0 for the similar pixels while the different pixels has the value 255. Figure 3.7 shows the difference between the images, where pixel value 255 is white in the grayscale level. Figure 3.7a and Figure 3.7b shows comparison between an Ok and an erroneous image where the bag clearly can be seen as large white patch. Figure 3.7c and Figure 3.7d are comparisons between two different OK images, which has some differences caused by shadows and minor offsets in the image, which can be seen as white patches in the images, but not as large as for the images with a bag in the ROI.



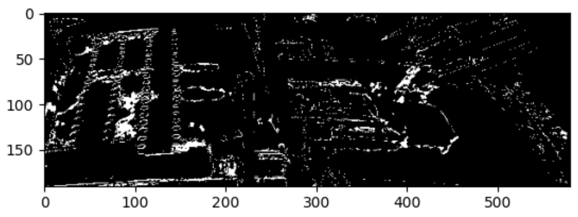
(a) Good vs. Bad example 1.



(b) Good vs. Bad example 2.



(c) Good vs. Good example 1.



(d) Good vs. Good example 2.

Figure 3.7: Examples of the comparison between images by subtracting the pixel values of one image from another

From this MSE has been chosen as the calculation method, since this method is quite similar to what the visual comparison did in Figure 3.7. MSE will then take the sum of each pixel error/between the target image and input image squared, giving a real number that increases as the difference increases. Although RMSE and PSNR were used earlier, these methods were not used further. Both methods are based on MSE. RMSE is the root of MSE and doesn't penalize differences as much as MSE. PSNR in addition to other methods are comparing similar images with respect to degrading, noise, blur and lighting conditions.

3.2.3 Detecting bags

Finally, a threshold value has been chosen to separate the images from the MSE values. This has been done with a trial and error from the MSE of 36 input images compared with target image. The algorithm has been tested with single and multiple target images. For the bag detection with multiple target images, the target images has been selected through trial and error to find target images with as many variations within the "Good" category as possible to allow for small variations in the input images. The final implemented algorithm's process is shown in the flowchart in Figure 3.8 where the target images and the input images are loaded, then all the images are going through the same preprocessing steps, rotate, crop, grayscale then Otsu thresholding. Afterwards, the difference between the target images and the input image is calculated with MSE. The average of all the MSE values for each target image is compared with a threshold value. If the MSE is larger than the threshold a bag is detected and the input image is saved to the folder “Bad”, if the MSE is below the threshold the image doesn’t have a bag and is saved to the folder “Good”.

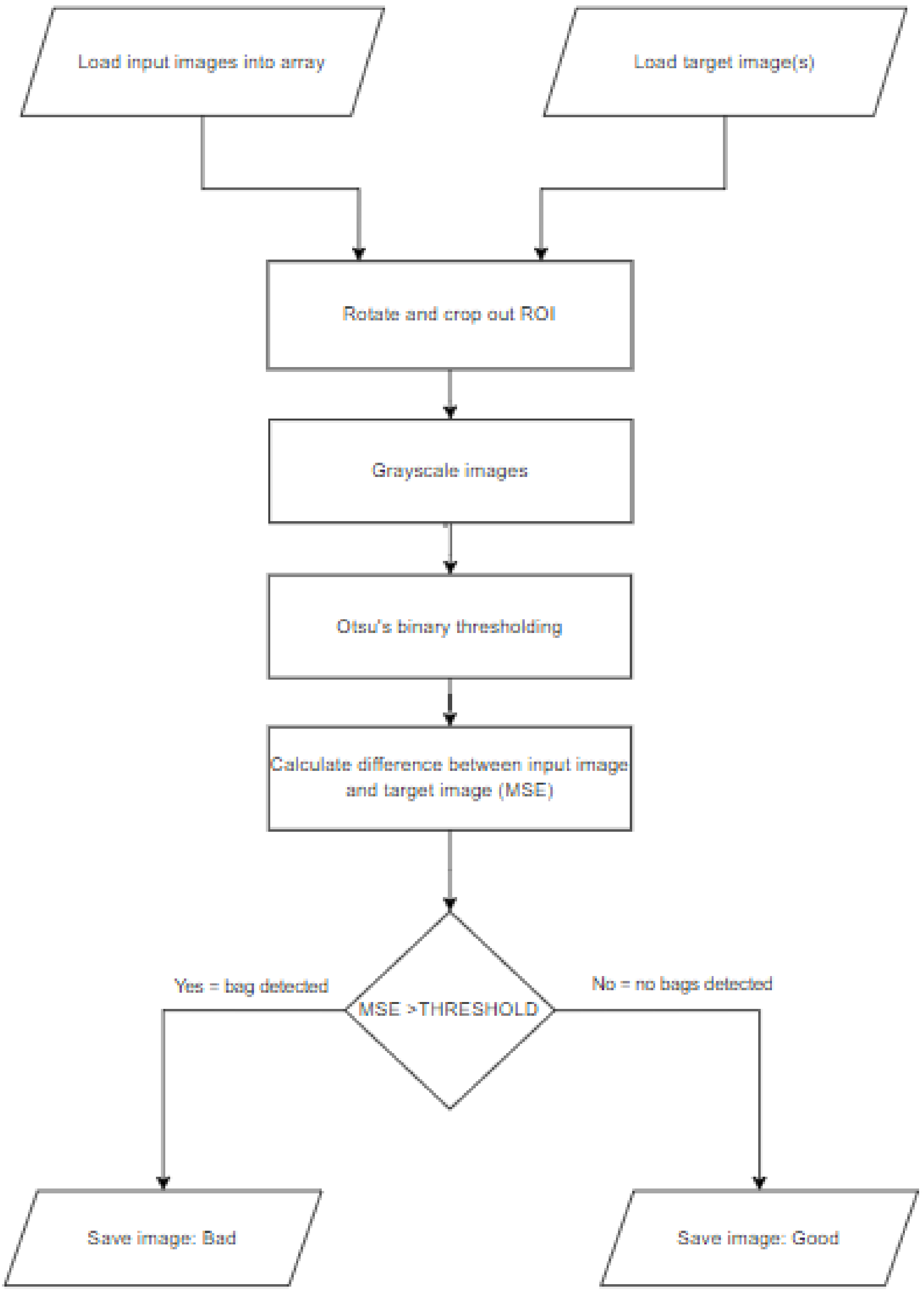


Figure 3.8: Flowchart of Model 1: bag detection algorithm.

3.3 Model 2: Anomaly detection using convolutional neural network

The CNN consists of three main parts shown in Figure 3.9. First the images are load processed and shuffled in (1). Then the model is trained (2) and finally we can use or test the model on data to predict an output (3). To develop the CNN model, Keras and TensorFlow has been used.

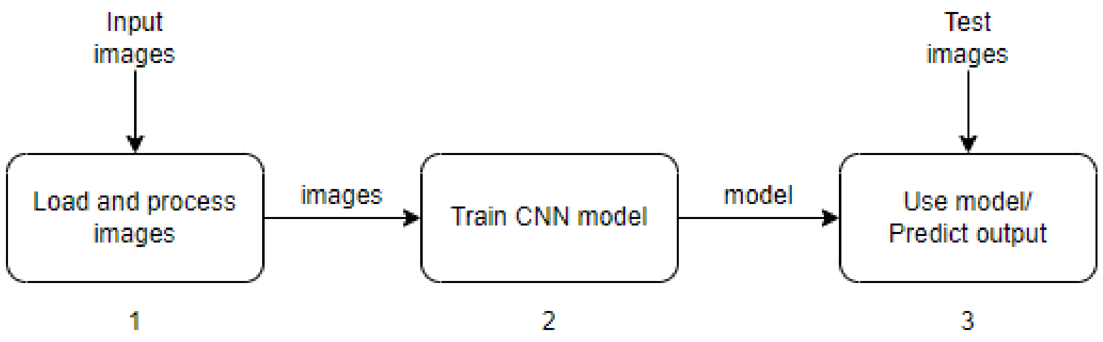


Figure 3.9: Flowchart of Model 2: anomaly detection using CNN.

3.3.1 Load and process images

To create the dataset ready to use for the CNN algorithm we load images from two folders. The names of the folders will be used as labels for the input images which separates the classes. For this model, it has been chosen to use the class ”Bad” for the images that includes anomalies and errors that is undesirable and ”Good” for the images of the cases that are desirable or prudent. The images are imported as grayscale to try to neglect the colored light from the TCP, this is because the color shouldn’t be a feature that separates the classes. In addition, this reduces the depth (number color channels) in the image, which reduces computational complexity of the model. The ROI is chosen to be pixels 90 to 460 in the y-axis and 525 to 790 in the x-axis of the original image, this is illustrated with the rectangular shape in Figure 3.10. The height (y-axis) of the ROI is larger than the case, this makes it possible to use data when the case is in different positions, and still get the entire case inside the ROI.

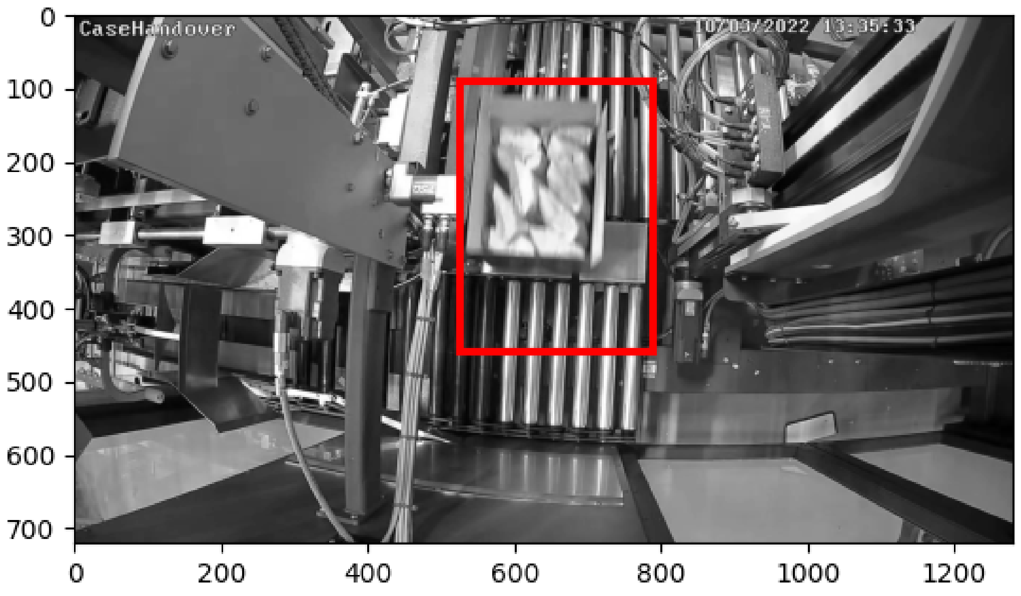


Figure 3.10: ROI for the input data to the CNN marked with a red rectangle.

Image augmentation

Image augmentation has been applied to artificially expand the size of the original data set. For this project the rotation function previously created in Section 3.2 for Model 1 has been applied in addition to simply cropping the image with small variations.

3.3.2 Training, validating and testing the CNN

Training

The next step is to train the CNN. A couple of different architectures and hyperparameters have been tested on different datasets and the results are presented in Section 4.2. The hyperparameters has to be selected before training. The hyperparameters for convolutional neural networks are shown in Table 3.2.

Table 3.2: List of hyperparameters to be set before training the CNN [10].

Layer	Parameters	Hyperparameters
Convolutional layer	Kernels	Kernel size, number of kernels, stride, padding and activation functions
Pooling layer	None	Poling method, kernel size, stride and padding
Fully connected layer	Weights	Number of neurons and activation function
Other		Model architecture, optimizer, learning rate, loss function, batch size, epochs and dataset splitting

The CNN has some general settings which are commonly used for image classification, and are used in this project as well, many of which are the best suited methods for binary classification. The architecture of the CNN consists of different number of layers. Keras' sequential model is used create the model, which is a feed forward neural network with a linear stack of layers. The Architecture is built up with three stacks of convolutional layers with ReLU activation function followed by a max pooling layer. The convolutional layers extracts the features in the image while max pooling reduces the dimensions of the feature maps. After these layers we flatten the feature map into one dimension, which gives a column like feature map. The 1D feature map is passed through a fully connected layer that maps the features and then are finally connected to the output fully connected layer with 1 neuron giving a classification of 0 or 1. The hyperparameters that has been used are shown in Table 3.3. Binary cross entropy is the most common loss function for binary classification problems that calculate the average difference between the actual and the predicted probability distribution. This function returns a value between 1 and 0, where 0 is perfect and 1 is worst. For optimizer Adam has been used. It is an adaptive optimizer that doesn't require a learning rate as input, whereas instead it maintains a learning rate for each network weight and adapts throughout the training. This method is easy to implement, works great on sparse data, and is computational effective [13].

Table 3.3: CNN model configuration and hyperparameters used to train the model.

Layer	Hyperparameter	Setting
Convolutional Layers	Kernel size	3x3
	Number of kernels	64
	Stride	1
	Padding	Zero-padding
	Activation function	ReLU
Pooling Layer	Method	Max Pooling
	kernel size (filter size)	2x2
	Stride	2
Fully Connected Layer	Neurons	64
Output Layer (fully connected)	Neurons	1
	Activation function	Sigmoid
Others	Optimizer	Adam
	Loss function	Binary cross entropy
	Dataset split	30% validation data
	Epochs	Varying (early stop)
	Batch size	32

Validation

For validation, the entire dataset has been split into 30% and 70%, where 70% of the data is used to train the model, and 30% is used for validation. The loss function calculate a loss that indicates how well the model is fit to the training data. And a validation loss indicates the model fit compared to the validation data. In addition to the loss, an accuracy is calculated, which indicates how accurate the model is able to classify the images compared to the actual output.

Testing

When the model is trained we can use test data to test the output prediction. The test dataset has to be unique to the dataset used when training and validating the model. The test data is fed through the same preprocessing steps that have been used on the training data. Then the test image is passed through the model with the predict function of the model. The output will be a value between 0 and 1 because of the final dense layer with a single neuron. The return value 0 is classified as ”Good” and the value 1 is classified as ”Bad”. The values in between gives an indication of how certain the model is of its output. For instance, if the output is around 0.5 the model aren’t able to distinguish between the two classes. The test data used to test Model 2 are 12 different images, which includes includes ”Bad” images, ”Good” images and a couple of images where the the arrangement are bad but there is no anomalies or errors in it. One of the images used for testing is quite blurry and badly arranged, this image was labeled as ”Bad”.

4 Results

The results of the two models will be presented in this chapter. Since the output of each test is based on input images, there has been selected a hand full of input images spanning from ”Good” to ”Bad” to showcase the entire range of the results for both models. The results of Model 1 will be presented first, followed by the results of Model 2.

4.1 Model 1: Bag detection

The image comparison algorithm was first tested with a single target image in subsection 4.1.1. And then, it also was tested with multiple target images, see subsection 4.1.2. For the image comparison with multiple target images, the averaged MSE was used to compare against the given threshold. 36 images are selected to test Model 1. These images has been chosen carefully to cover all differences caused by light and shadows.

4.1.1 Results - using a single target image

These results are of image comparison with a single target image. The threshold separating the MSE was selected as 5000. This was tested with many different target images, but the results of two cases are shown as examples here. The first results of Model 1 using single target image used Figure 4.1 for similarity comparison.

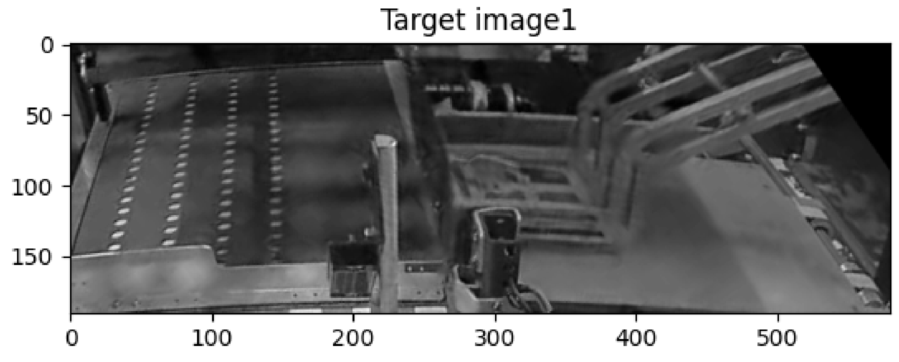
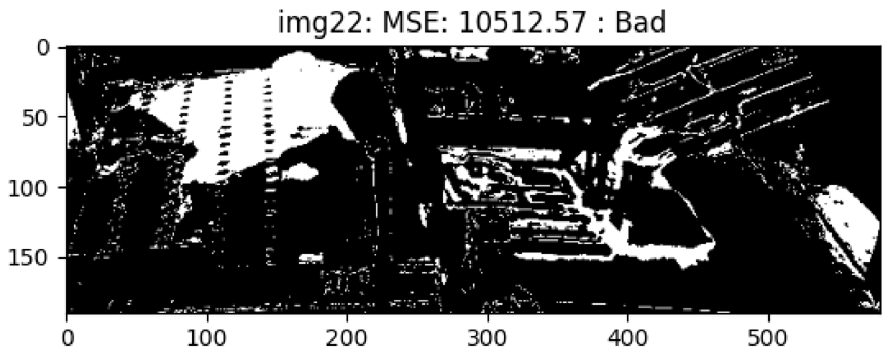
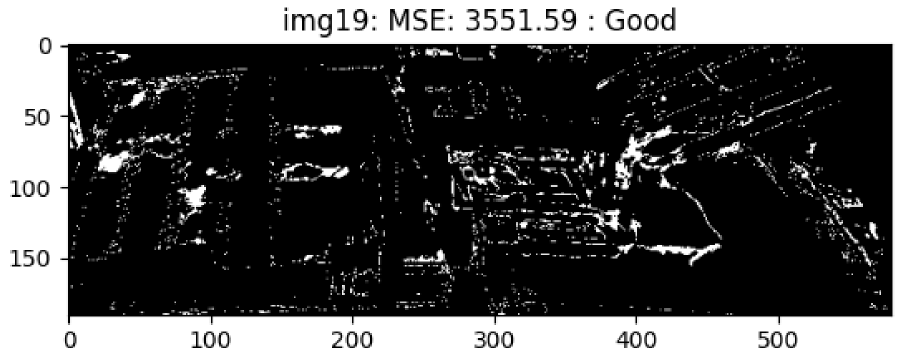


Figure 4.1: Target image 1, used to compare with the input images for Model 1.

From the obtained results, this model is able to separate 33 of the 36 images correctly. Figure 4.2a shows an example result of an image classified as ”Bad” where the MSE is 10512.57, which is much higher than the threshold value 5000. This image has a two bags that failed to get pushed inside the case, one can clearly be seen at the left of the center, and the other one is all the way to the right, which is not as visible. Figure 4.2b is an example result of an image classified as ”Good” where there are no bag inside the frame, making the difference image quite dark. The MSE is 3551.59 that comes from small differences between the input image and the target image.



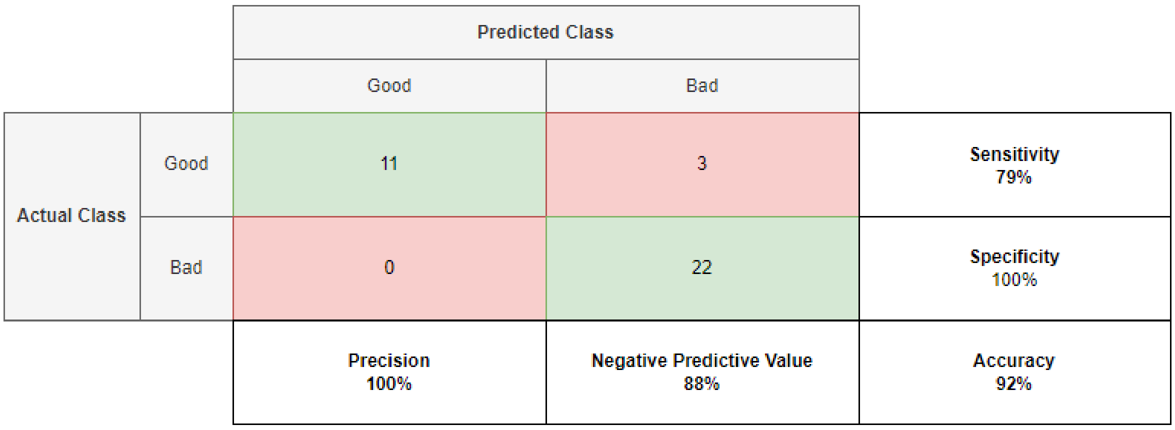
(a) Example of a correct classification of a ”Bad” image that contains a bag for Model 1 with single target image.



(b) Example of a correct classification of a ”Good” image without a bag for Model 1 with single target image.

Figure 4.2: Example of output from image comparison with target image given in Figure 4.1.

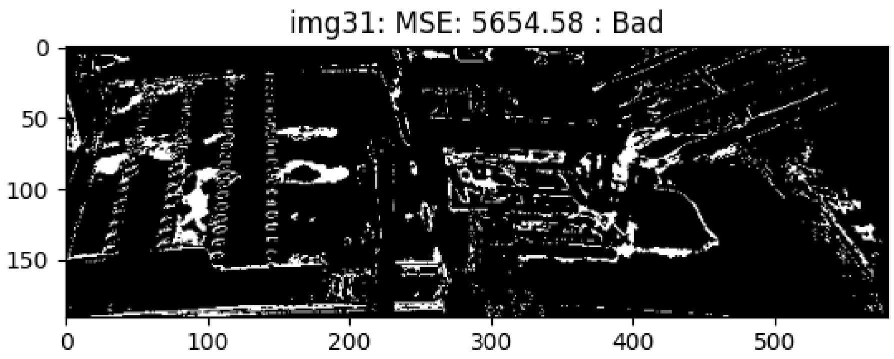
However, not all the test images gave good results, this is reflected in the confusion matrix in Figure 4.3. The confusion matrix tells us that that the precision of the predictions are 100%, which means that every ”Good” prediction is correct. However, since some of the ”Good” images are classified as ”Bad” the sensitivity and negative predictive value is reduced. This means that the model aren’t able to predict every ”Good” image and we can’t be certain that the ”Bad” predictions are correct. The overall accuracy of the model is 92%.



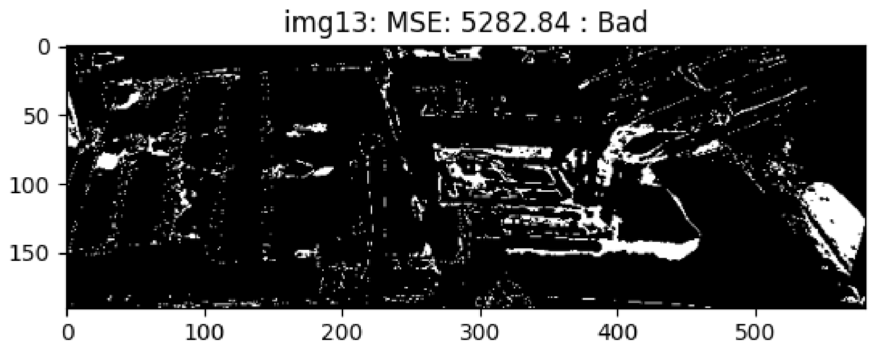
		Predicted Class		
		Good	Bad	
Actual Class	Good	11	3	**Sensitivity** 79%
	Bad	0	22	**Specificity** 100%
		Precision 100%	**Negative Predictive Value** 88%	**Accuracy** 92%

Figure 4.3: Confusion matrix of the model predictions obtained with the single target image shown in Figure 4.1.

Figure 4.4a shows a example of a wrongly predicted ”Good” image. This image has no bag in it, which should have been classified as ”Good”. The MSE of this image is 5654.58 which is quite higher than the threshold on 5000. It is neither possible to increase the threshold to make this image be classified as ”Good” because it would lead to missclassification of the ”Bad” image in Figure 4.4b with 5282.84 MSE.



(a) Example of a wrong classification of a actual ”Good” image, classified as ”Bad”.



(b) Example of a correct classification of a actual ”Bad” image, but here the MSE is below the actual good classification in Figure 4.4a.

Figure 4.4: Example of output from image comparison with target image given in Figure 4.1.

The majority of the results with single target image the model could not separation all the ”Good” images from the ”Bad” images where the bag was barely visible, except for the test done with Figure 4.5 as target image. The MSE values closest to the Threshold value are shown in Figure 4.6, where the MSE of the ”Good” image is 4957.71 and the MSE of the ”Bad” image is 5617.40. The remaining 34 test images’ MSE values was further from the threshold. The good result obtained with Figure 4.5 as target image are reflected in the confusion matrix shown in Figure 4.7, with an accuracy of 100%.

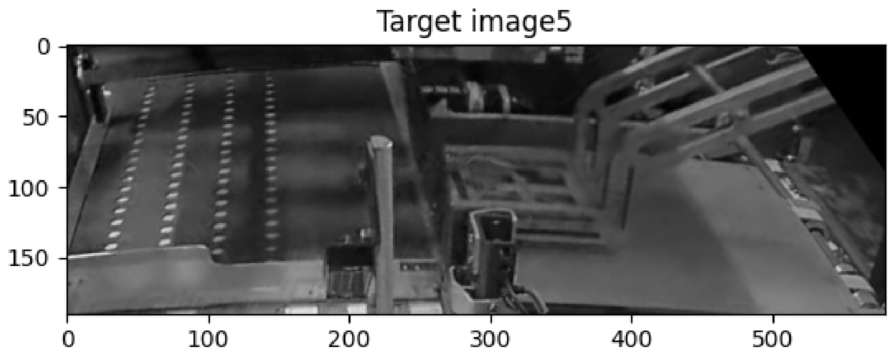
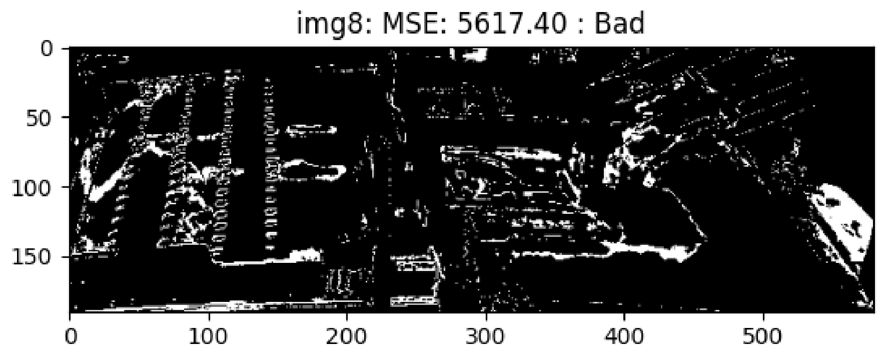
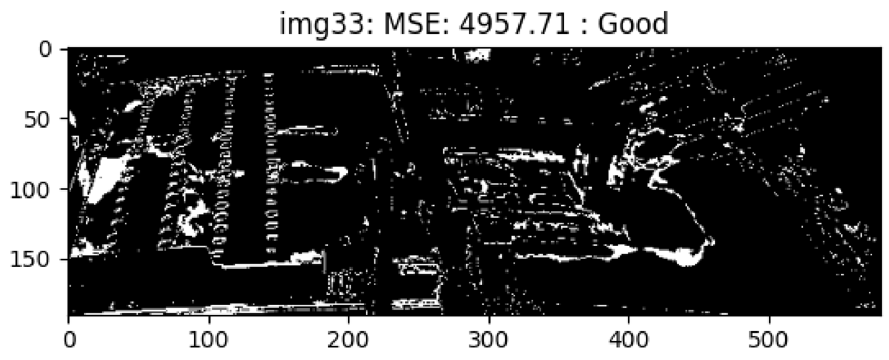


Figure 4.5: Target image 5, used to compare input images for Model 1.

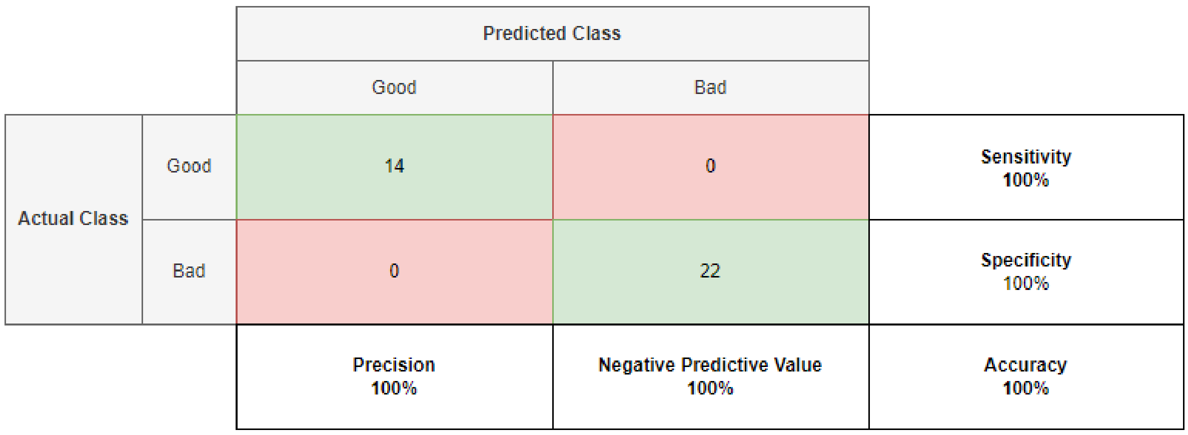


(a) Example of a correct classification of a ”Bad” image that contains a bag for Model 1 with single target image



(b) Example of a correct classification of a ”Good” image without a bag for Model 1 with single target image.

Figure 4.6: Example of output from image comparison with target image given in Figure 4.5

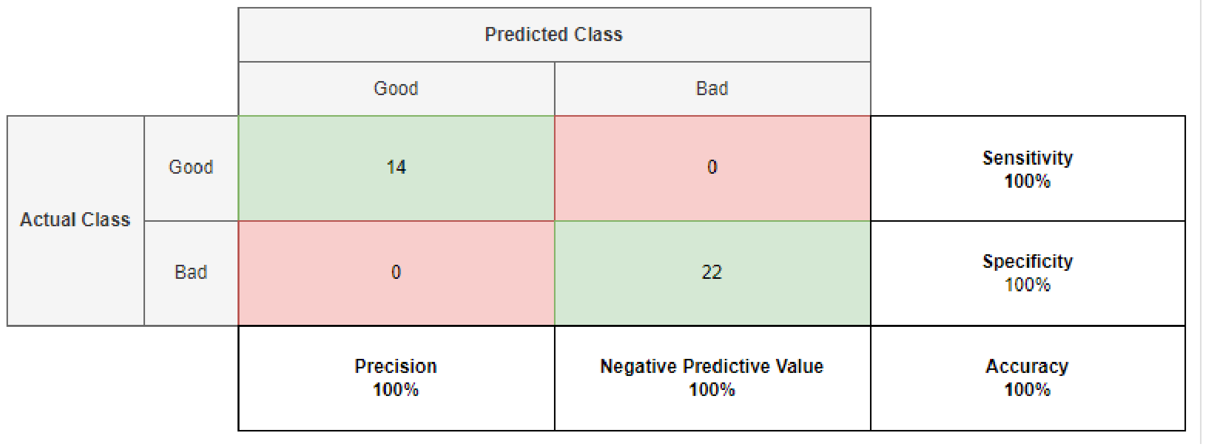


		Predicted Class		
		Good	Bad	
Actual Class	Good	14	0	**Sensitivity** 100%
	Bad	0	22	**Specificity** 100%
		Precision 100%	**Negative Predictive Value** 100%	**Accuracy** 100%

Figure 4.7: Confusion matrix of the model predictions obtained with the single target image shown in Figure 4.5.

4.1.2 Result - using multiple target images

This subsection shows the results of the image comparison algorithm for bag detection with 6 different target images, using the average of the MSE between each target image to compare against a threshold value. The threshold separating the ”Good” and ”Bad” for this test was selected as 4900, as the highest value for the good images was 4814.13 and the lowest for the ”Bad” images was 5059.60. The output of the model separated all the 36 test images correctly. This shown in Figure 4.8, with 100% accuracy.



		Predicted Class		
		Good	Bad	
Actual Class	Good	14	0	**Sensitivity** **100%**
	Bad	0	22	**Specificity** 100%
		Precision **100%**	**Negative Predictive Value** **100%**	**Accuracy** **100%**

Figure 4.8: Confusion matrix of the model predictions obtained with the single target image s

4.2 Model 2: Anomaly detection using CNN

The main results from Model 2 are presented in this section. The hyperparameters selected for Model 2 are shown in subsubsection 3.3.2. This section is separated into two. First the results from the model trained without image augmentation. This model was trained using a collection of all the erroneous data from one recipe for the TCP. When training the algorithm, the non-erroneous data was randomly selected to get comparable data for both classes, ”Good” and ”Bad”. The second model results are obtained from a model trained on a different dataset. Some of the data was removed as it was hard to put some of it into a category even with human vision. This was caused mainly by the TCP badly arranging/pushing the products inside the case and blurring due to motion. In addition, augmentation was added to increase the dataset size. Early stop callback function was also added to stop the training when the validation loss don't improve for two epochs and then the best weights are restored. When testing both these CNN models, 12 images has been selected to make output predictions. The predicted classes are also compared to the actual classes in a confusion matrix for bot models.

4.2.1 Results of the model trained without image augmentation

Figure 4.9 and Figure 4.10 shows the accuracy and loss of the model. The loss and accuracy is represented in the y-axis, while the x-axis is the number of epochs. val_accuracy and val_loss shows the accuracy and loss based on the validation data. Based on the resulting low accuracy and the high loss when training the model, it is not well trained. The accuracy should be closer to 1 and the loss should be closer to 0 in a well trained CNN. The bad results are also reflected on the prediction output shown in Table 4.1. Although images of two categories, ”Good” and ”Bad”, were used in this test, all predicted values were above 0.5, which means that all images were classified as ”Bad”. The predictions are very close to 0.5, which means the model are not certain about its classification. From the confusion matrix in Figure 4.11 it is shown that the accuracy is low. The only classification that can be trusted is that the ”Bad” test data will be classified as ”Bad”.

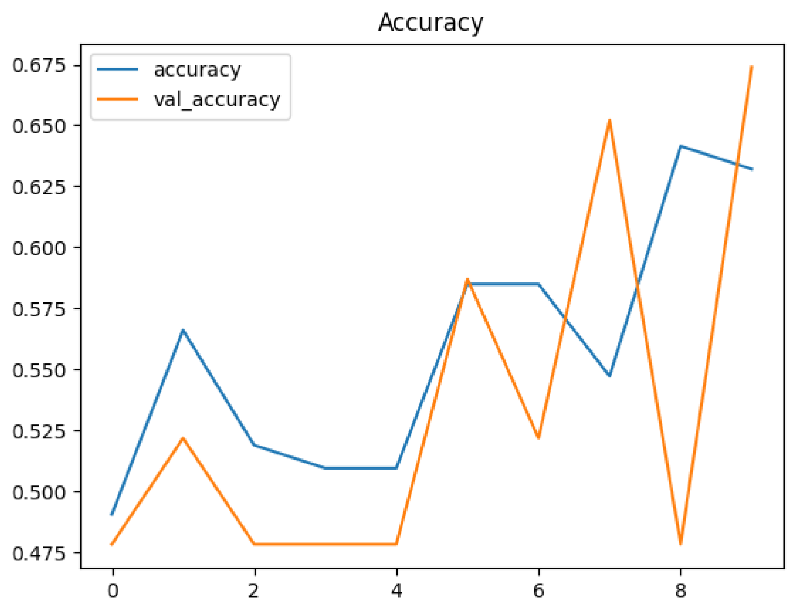


Figure 4.9: Accuracy of the CNN trained without image augmentation. The x-axis is the number of epochs.

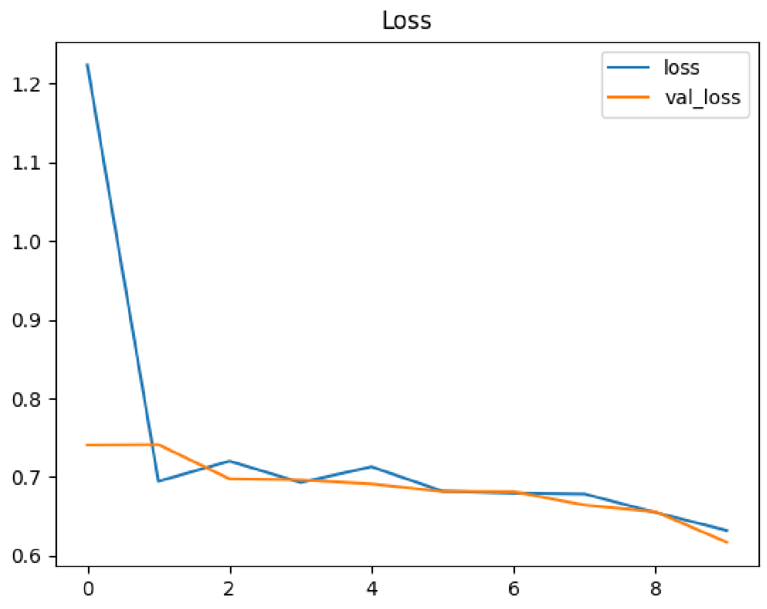
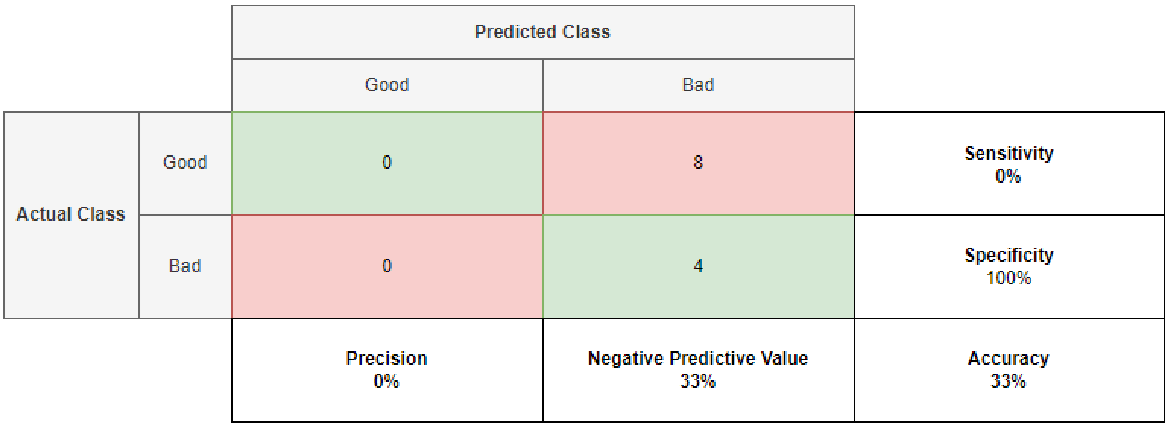


Figure 4.10: loss of the CNN trained without image augmentation. The x-axis is the number of epochs.

Table 4.1: Predicted output for CNN trained without image augmentation.

Images	Prediction
Image 1	0.5673915
Image 2	0.6110512
Image 3	0.6190277
Image 4	0.73876935
Image 5	0.6433699
Image 6	0.6223888
Image 7	0.61720955
Image 8	0.54970825
Image 9	0.70532084
Image 10	0.6027814
Image 11	0.62704635
Image 12	0.56894445



		Predicted Class		
		Good	Bad	
Actual Class	Good	0	8	Sensitivity 0%
	Bad	0	4	Specificity 100%
		Precision 0%	Negative Predictive Value 33%	Accuracy 33%

Figure 4.11: Confusion matrix of the model predictions obtained from the CNN trained without image augmentation.

4.2.2 Results of the model trained with image augmentation

The accuracy and validation accuracy of the CNN is shown in Figure 4.12, and the loss and validation loss is shown in Figure 4.13. The validation loss stopped improving after the 7th epoch, if we count the 0th epoch. This made the weights used for the stored model give an accuracy of 0.9797 and validation accuracy 0.9588.

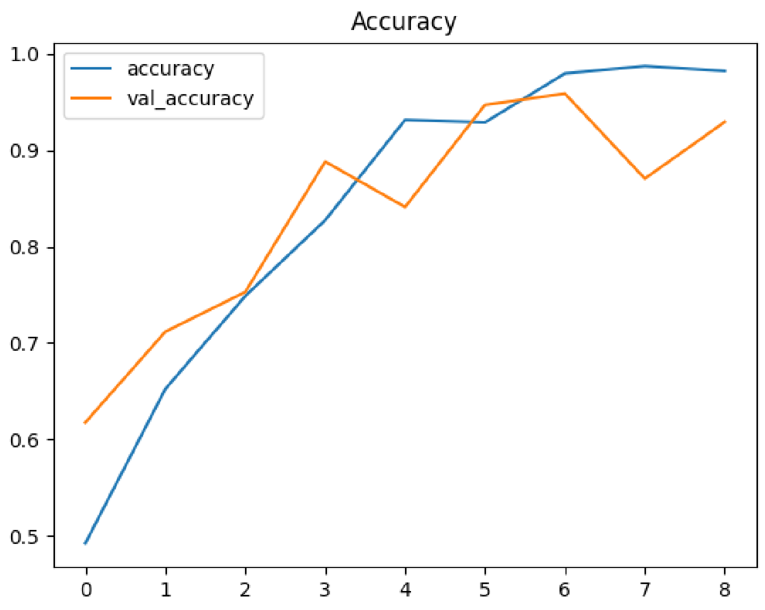


Figure 4.12: Accuracy of the CNN trained with image augmentation. The x-axis is the number of epochs.

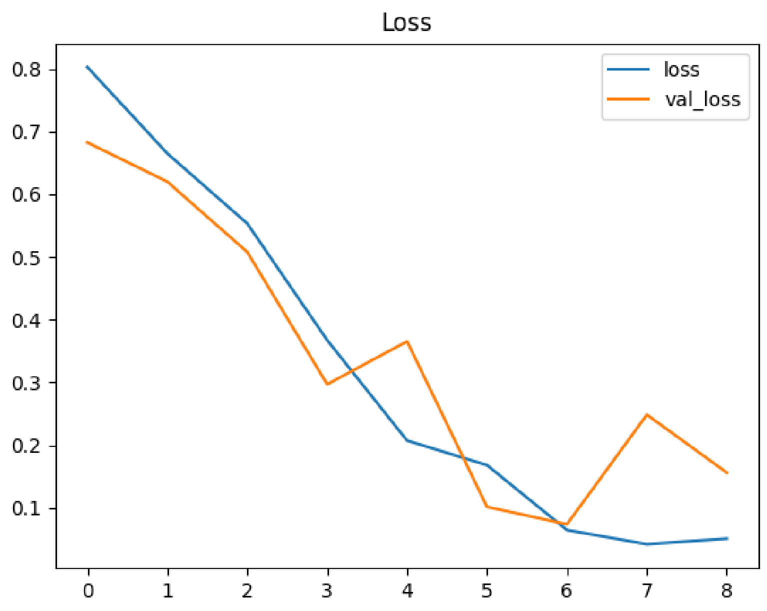
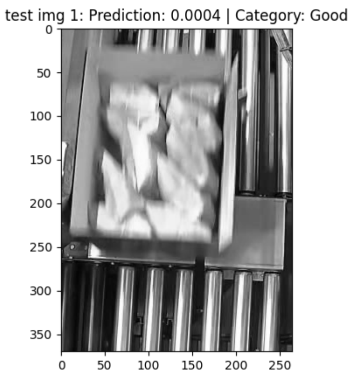
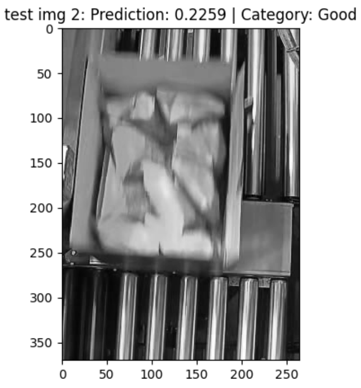


Figure 4.13: loss of the CNN trained with image augmentation. The x-axis is the number of epochs.

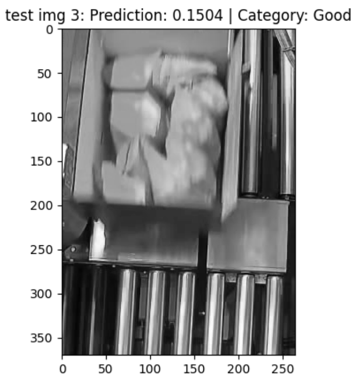
The model was further tested by using the test dataset, the output predictions are shown in Figure 4.14. With this CNN all the predictions are correct compared to the actual class. The predicted output is printed in the label of each image with the category of the related class. The output prediction of the test images shown in Figure 4.14a, Figure 4.14g and Figure 4.14l are almost 0, giving classification of the images as ”Good” with high certainty. The output shown in Figure 4.14d, Figure 4.14e and Figure 4.14i shows that the network is able to predict the ”Bad” cases with high certainty. The remaining images is correctly classified as either ”Good” or ”Bad” but the certainty of the model is not as high. The confusion matrix is shown in Figure 4.15, this reflects that all images was classified correctly and the prediction accuracy of the testdata is 100%.



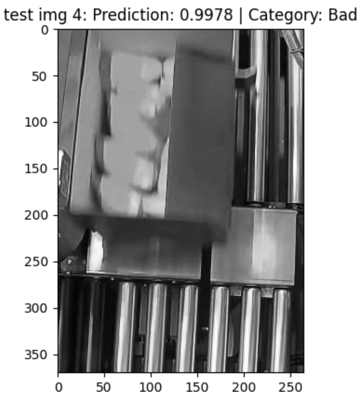
(a)



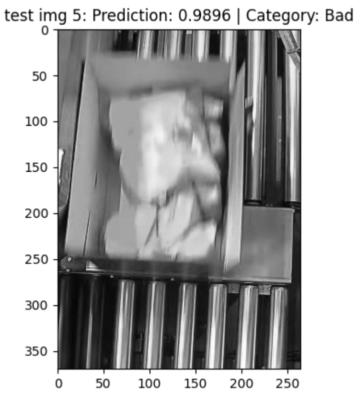
(b)



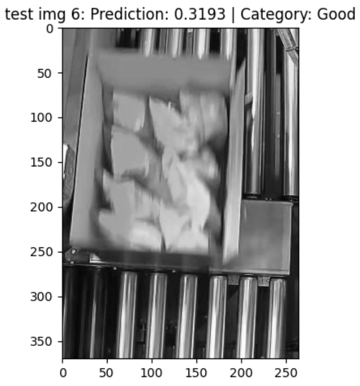
(c)



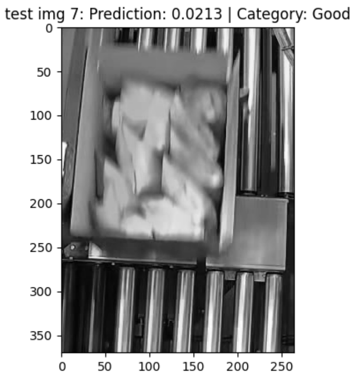
(d)



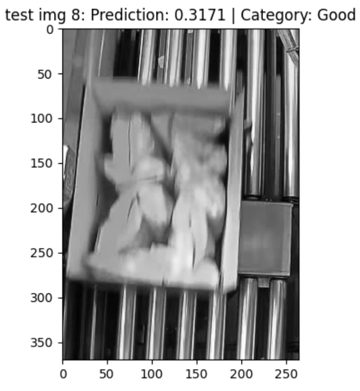
(e)



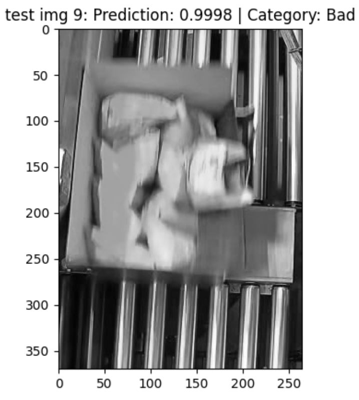
(f)



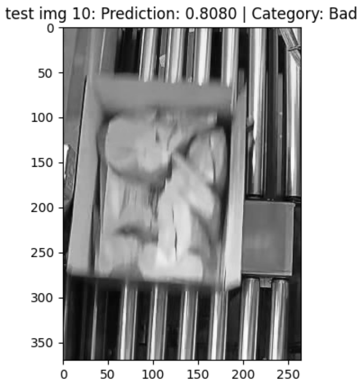
(g)



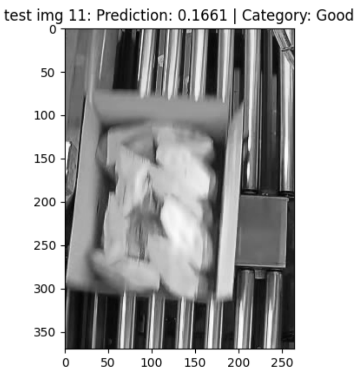
(h)



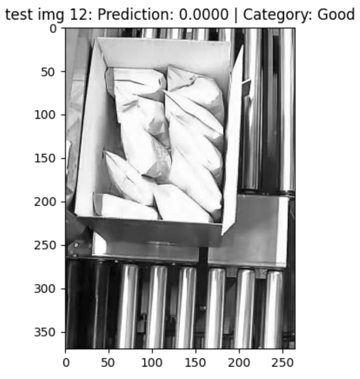
(i)



(j)

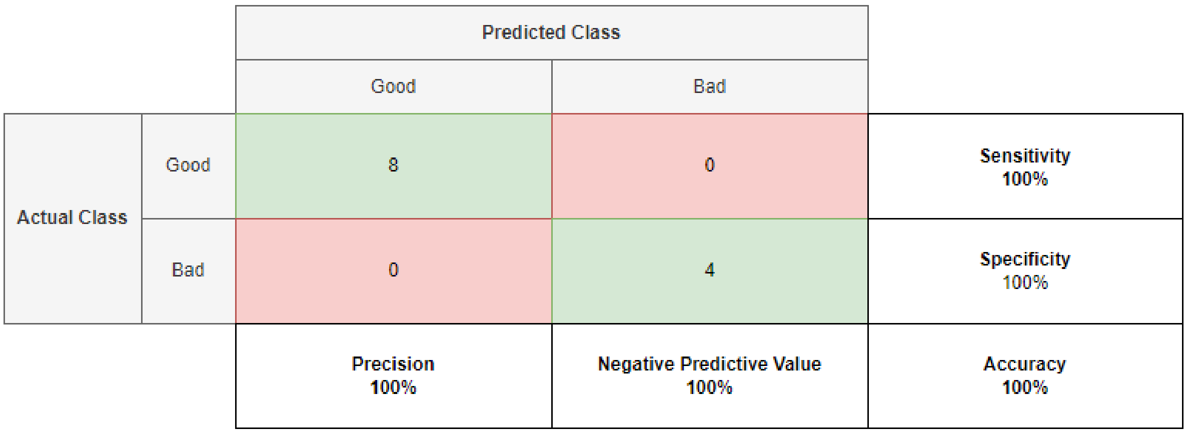


(k)



(l)

Figure 4.14: Prediction output of the Model 2. There are 12 test images fed into the model and the output prediction are printed in the label of each image.



		Predicted Class		
		Good	Bad	
Actual Class	Good	8	0	**Sensitivity** **100%**
	Bad	0	4	**Specificity** 100%
		Precision **100%**	**Negative Predictive Value** **100%**	**Accuracy** **100%**

Figure 4.15: Confusion matrix of the model predictions obtained from the CNN trained with image augmentation.

5 Discussion

This chapter primarily discusses the results of the two models, and what could have been done to improve the models. This research consists of collecting data and training two models. Collecting data is important, and gives a basis for developing the models. In this project, the data was collected during commissioning when the machine was newly installed in Belgium. The machine hasn't been adjusted optimally at this stage, and erroneous situations occur more frequently. In addition, the data was collected on flipped products which is more difficult to run optimally. This seemed to be a good solution when collecting data for the models to be able to get a lot of data of ”Bad” situations, but the data collected of images without errors wasn't representing how it normally looks when the machine is fine tuned. The bags were badly arranged inside the case and on top of this the collected data had some blur due to motion. This made it hard to put a lot of the data into the category for ”Good and ”Bad”.

5.1 Model 1: Bag detection

The hardest part with the bag detection model was to include enough target images and correct images to distinguish the images with a bag located all the way on the right as in Figure 4.6a from the images without a bag that had some differences due to light and other variations as shown in Figure 4.6b. We were able to separate these two with the model that only used one target image and for the one with multiple target images. The difference in MSE value between the two images closest to the threshold was less for the model using a single target image than for the model using multiple target images. However, the model using a single target image that worked, was 1 out of 6 images that was tested as target image, and this Model would probably be worse when an image with a new variation occurs.

The ”Bad” results with the images that had a bag on the right side from Figure 4.6a was a following error from a bag that was not removed after a previous error. As this bag was detected when it first occurred the model could be implemented to the system giving an alarm with acknowledgement requirement of removing the bag before restarting the TCP. Suppose we can neglect these situations where the bag has already been detected and fallen halfway down in the back of the machine. In that case, it is possible to increase the threshold value quite a lot before the model wrongly will classify a detected bag as ”Good”. This will again allow for bigger variations in the images without a bag.

5.2 Model 2: Anomaly detection using CNN

The CNN model that was trained on the dataset without image augmentation to increase the dataset size when training the model, gave bad results. This is a typical case of a underfitted model. The data used to train the model was insufficient to accurately capture relationship between the features of the input data and the output classes. The underfitting is the reason for the bad accuracy and loss given in subsection 4.2.1.

To solve this problem, a new model was trained with image augmentation. This model gave high accuracy and the loss was low. This was also reflected in the output predictions shown in Figure 4.14. Although, the confusion matrix for this model's outputs gave 100% accuracy shown in figure Figure 4.15, there is still room for improvements to increase the certainty of the model when giving a predicted output. In addition, the data classified as ”Bad” had somewhat similar anomalies with the data it was trained on, and it is hard to say for certain that the model will give as good results when there is a new unique situation with anomalies/errors. Generally, more data should have been included.

6 Conclusion

The main goal of this thesis was to develop two different models using two different principles in computer vision. Model 1: Bag detection was developed using image comparison methods. The final model that used multiple target images for similarity comparison are able to detect bags that are not correctly pushed inside the case. Model 2: anomaly detection using CNN, was developed using a convolutional neural network. This model also gave good results based on the limited data collected of anomalies, when the dataset size was increased with image augmentation. However, it was desirable to create a model that works for all recipe settings on the TCP. This was not achieved with this model.

During the execution of this project, due to limited time, it was decided to shift the development focus away from the live implementation of the model. Moreover, the implementation and deployment of the model are quite tedious tasks as a working model is needed before the deployment. Thus, it was decided to spend more time on developing a robust model with high performance. Consequently, it was not implemented the tasks about data storing, live implementation, deployment and evaluating the feasibility of keeping the developed models in one system. It is a shortcoming in this project. However, the possible solutions to implement the models are suggested below.

6.1 Suggestions for further work

To implement the models the simplest way would be to use a computer connected to the PLC and the camera. Since both models use images as input, the data collection algorithm should use the camera to take snapshots directly fed into the models instead of storing the data. With the connection to the PLC we can get signals to trigger a camera snapshot. For Model 1 this would be after each push from the pusher, a Boolean signal could be set true based on the pusher position. The TCP already has a photocell that detects the full case in the Full Case Transfer EM, this could be used to give a signal to Model 2.

The final CNN model was trained on data from one recipe, meaning one specific type of bag, packed inside one specific type of case. This means that the current model can't classify anomalies/errors on every possible recipe option running on the TCP. Anyhow, this CNN shows very promising results that it is possible to use CNN to detect anomalies and can be implemented to run for this recipe. To further improve the model to be more generalized for all recipe options two suggestions are given below:

- Use data from different recipes to train a generalized model that works for all of them.
- Train one model for each recipe and select the model to classify data according to the active recipe.

Training one general CNN model could be difficult because of big differences in case size, patterns inside the case, bag size and bag color. And training multiple models can be tedious and not very generalized, a combination of the two suggested solutions could be the best way to solve this, by grouping similar recipes to train a couple of CNN models that cover all recipes.

Both models are computationally effective and fast. The models can easily be ran on one computer without a GPU, as the models only will process one image at a time. If it is possible to train one general CNN model for all recipes, it could also be possible to interconnect many machines to a single CNN model, as it is normal for a CNN model to compute output predictions on a large input tensor, which can contain many input images.

Bibliography

[1] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning,' *Nature*, vol. 521, no. 7553, p. 444, 2015.

[2] R. Poudel and A. L. Lemme, 'Anomaly detection and maintenance for chips packaging machines using machine learning,' FM4017 Project 2021, University of South-Eastern Norway, Porsgrunn, 2021.

[3] A. Rao and R. Lanphier, 'Real Time Streaming Protocol(RTSP),' Internet Engineering Task Force, Internet-Draft draft-rao-rtsp-00, Oct. 1996, Work in Progress, 43 pp. [Online]. Available: `https://datatracker.ietf.org/doc/html/draft-rao-rtsp-00`.

[4] B. Posey. 'Real time streaming protocol (rtsp).' (12 March 2018), [Online]. Available: `https://www.techtarget.com/searchvirtualdesktop/definition/Real-Time-Streaming-Protocol-RTSP`.

[5] M. M. Mela, 'Utilizing dsp for ip telephony applications in mobile terminals,' Master's Thesis, Helsinki University of Technology, Helsinki, 2006.

[6] L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing: A Gentle Introduction Using Java.* Springer, 2017.

[7] N. Otsu, 'A threshold selection method from gray level histograms,' *IEEE Trans. Systems, Man and Cybernetics*, vol. 9, pp. 62–66, Mar. 1979, minimize inter class variance.

[8] K. O'Shea and R. Nash, 'An introduction to convolutional neural networks,' *CoRR*, vol. abs/1511.08458, 2015. arXiv: `1511.08458`. [Online]. Available: `http://arxiv.org/abs/1511.08458`.

[9] A. Ingargiola. 'Deep-dive into convolutional networks.' (Mar. 2019), [Online]. Available: `https://towardsdatascience.com/deep-dive-into-convolutional-networks-48db75969fdf`.

[10] R. Yamashita, M. Nishio, R. K. G. Do and K. Togashi, 'Convolutional neural networks: An overview and application in radiology,' *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[11] 'Machine learning.' (2014), [Online]. Available: `https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/`.

[12] 'Ai ml engineering.' (2020), [Online]. Available: https://www.debadityachakravorty.com/ai-ml/cmatrix/.

[13] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *arXiv preprint arXiv:1412.6980*, 2014.

[14] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks,' *Advances in neural information processing systems*, vol. 25, 2012.

[15] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, *Keras-tuner*, https://github.com/keras-team/keras-tuner, 2019.

Appendix A

Master’s thesis description

University of
South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Anomaly detection for Packaging Machines using Machine Learning

USN supervisor: Ru Yan, Ola Marius Lysaker

External partner: Tronrud Engineering

Task background:
Tronrud Engineering produces Tray and Case Packer machines (TCP) packing potato chips and other products into cardboard boxes. The third generation of TCP has been developed in the last couple of years, focusing on improving Overall Equipment Effectiveness (OEE). Today it has standard adjustments that can let the TCP suits various products for different customers. By doing this, the TCPs are enabled to be adjusted for different customers' requirements.
The TCP machines are equipped with different sensors and an Industrial Internet of Things solution for data collecting, transferring, and storage at a cloud-based platform, where the data will be stored, analyzed and visualized further.
Moreover, the TCP should be extended with capabilities for computer vision to detect anomalies during the operation and improve the product quality and better fulfil the customer's quality requirements.
This thesis is a continuation of the previous master project where the collected sound and vision data from the same TCP were analysed preliminarily.

Task description:
The overall task is to use machine learning techniques to improve the production quality for the TCP. According to the types of problems to be detected, the work is divided into two parts where two different models will be developed:

- Model 1 – implement an algorithm based on image processing to detect if any products fail to be pushed into the cardboard box.
- Model 2 – implementing a machine learning algorithm for computer vision to exam whether cardboard boxes have been appropriately filled.

These models should use images/video input from cameras. Using the methods for collecting and pre-processing vision data from the master project should be a starting point for developing models. In addition, it is necessary to evaluate the feasibility to keep the two models in one system or two separated systems and then implement the solutions

Subtasks:

1. Make a literature study on machine learning methods and relevant image processing topics.
2. Discuss and describe the process for collecting and storing data.
3. Investigate and choose optimal camera position and resolution for Model 1 and Model 2.
4. Describe the process of preparing data for the algorithms.
   4.1. Define suitable target image and similarity threshold for image matching.
   4.2. Explain training, validation and testing of the machine learning algorithm. Include considerations for under- and overfitting the model.

5. Study and propose any available applicable algorithms for these applications and describe the model structure including layers, neurons, epochs and activation functions that can be used.
6. Give an overview of methods for measuring accuracy and performance on such models.
7. Implement and test the algorithms on data from the TCP.
8. Validate the models with respect to accuracy and performance.
9. Deploy the model to connect with live data from the TCP and evaluate the performance.
10. Evaluate the feasibility to keep the two developed models in one system or two separated systems and then implement the solutions.

Student category: Reserved for Industry master student Anders Lysgaard Lemme.

The task is suitable for online students (not present at the campus): No

Practical arrangements:
The machine will be available at Tronrud Engineering, Hønefoss.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): 04.02.2022 Ru Yan

Student (write clearly in all capitalised letters): ~~A~~ ANDERS LYSGAARD LEMME

Student (date and signature): Anders Lysgaard Lemme, 04.02.2022