Vähä, Eemil

An obstacle detection system for automated guided vehicles

School of Technology and Innovations
Master's thesis in Automation and Computer Science
Energy and Information Technology

Vaasa 2022

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Author: Vähä, Eemil
Title of thesis: An obstacle detection system for automated guided vehicles: [Subject]
Degree: Master of technology
Subject: Automation and information technology
Supervisor: Jouni Lampinen
Graduation year: 2023 **Sivumäärä:** 112

Abstract:
The objective of this master's thesis is to investigate the utilization of computer vision and object detection as an integral part of an automated guided vehicle's navigation system, which operates within the facilities of the target company. The rationale for conducting this research and developing an application for this purpose arises from the inability of automated guided vehicles to detect smaller or partially obstructed objects, and the lack of differentiation between stationary and moving objects. These limitations pose a safety hazard and negatively impact the overall performance of the system. The anticipated outcome of this thesis is a proof-of-concept computer vision application that would enhance the automated guided vehicle's obstacle detection capacity. The primary aim is to offer practical insights to the target company regarding the practical implementation of computer vision by developing and training a YOLOv7 object detection model, as a proposed resolution to the research problem.

A thorough theoretical part of the required technologies and tools for training an object detection model is followed by a plan for the application to define requirements for the object detection model. The training and development are conducted using open-source and standard software tools and libraries. Python is the primary programming language employed throughout the development process and the object detector itself constitutes a YOLOv7 (You Only Look Once) object detection algorithm. The model is trained to identify and classify a predetermined number of objects or obstacles that impede the present automated guided vehicle system. Model optimization follows a fundamental trial-and-error methodology and simulated testing of the best-performing model. The data required for training the object detection model is obtained by attaching a camera to an automated guided vehicle and capturing its movements within the target company's facilities. The gathered data is annotated using Label studio, and all necessary data preparation and processing are carried out using plain Python.

The result of this master’s thesis was a proof of concept for a computer vision application that would improve and benefit the target company’s day-to-day operations in their production and storage facilities in Vaasa. The trained model was substantiated to perform up to expectations in terms of both speed and accuracy. This project not only demonstrated the application's benefits but also laid grounds for the business to further utilize machine learning and computer vision in other areas of their business regarding the operational improvement competency of the target company’s employees. The results of this master’s thesis showed that finding an optimal object detection model for a specific dataset within a reasonable timeframe requires both appropriate tools and sufficient research data premises in terms of model configuration. The trained model could be utilized as a foundation for similar projects and thereby reduce the time and costs involved in preliminary research efforts.

AVAINSANAT: Computer vision, object detection, deep learning, Convolutional Neural Networks, Automated guided vehicle

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Författare: Vähä, Eemil
Avhandlingens titel: An obstacle detection system for automated guided vehicles: [Subject]
Examen:
Subjekt: Energia ja informaatiotekniikka
Handledare: Jouni Lampinen
Examensår: 2023 **Antal sidor:** 112

Abstrakt:
Syftet med detta examensarbete är att undersöka och använda datorsyn och objektdetektering som en del av navigationssystemet för en automatisk gaffeltruck som funktionerar i målföretagets lokaler. Motivationen för att skriva denna examensarbete och utveckla applikationen är den dåliga förmågan av den nuvarande automatiska gaffeltrucksystemet att upptäcka mindre eller på annat sätt vaga föremål och hinder. Automatiska gaffeltruckar klarar inte heller av att skilja rörliga föremål från stationära, vilket medför en klar säkerhetsrisk och försvagar systemets prestanda. Slutresultatet av detta arbete förväntas bli ett proof of concept för en datorvisionsapplikation, vilket potentiellt kan förbättra automatiska gaffeltruckens förmåga att upptäcka hinder i sin väg. Det primära syftet är att erbjuda praktiska insikter till målföretaget angående den praktiska implementeringen av datorsyn genom att utveckla och träna en YOLOv7-objektdetekteringsmodell, som ett förslag till lösning på forskningsproblemet.

Början av detta examensarbete innehåller en teoretisk del av de teknologier och tekniker som krävs för att utveckla en objektdetekteringsmodell, följt av en plan för tillämpningen för att definiera kraven. Öppna standardverktyg och databibliotek som Jupyter och OpenCV används för att utveckla objektdetekteringsmodellen och bearbeta data. Objektdetekteringsmodellen använder YOLOv7-algoritmen, som tränas för att upptäcka och identifiera definierade hinder som orsakar problem för det nuvarande systemet. Modellens optimering utförs genom att prova olika konfigurationer, varefter modellen med bäst prestanda testas i en simulerad miljö. Den data som behövs för att utveckla modellen för objektdetektering samlas in genom att fästa en kamera på en automatisk gaffeltruck och filmar dess dagliga verksamhet i målföretagets lokaler. Programvaran Label Studio används för att anteckna objekten, och all nödvändig dataförberedelse och bearbetning utförs med Python-programmeringsspråket.

Resultatet av detta examensarbete var ett proof of concept för en datorvisionsapplikation som skulle förbättra och gynna målföretagets dagliga verksamhet i deras produktions- och lagringsanläggningar. Den tränade modellen var kapabel att uppnå förväntningarna både vad gäller hastighet och noggrannhet. Detta examensarbete visade inte bara applikationens fördelar utan lade också grunden för målföretaget att ytterligare använda maskininlärning och datorsyn i andra delar av sin verksamhet genom att förbättra de operativa kompetensen hos målföretagets anställda. Resultaten av detta examensarbete visade att för att hitta en optimal objektdetekteringsmodell för en specifik datauppsättning inom en rimlig tidsram krävs både lämpliga verktyg och tillräckliga forskningsdatapremisser. Den tränade modellen skulle kunna användas som en grund för liknande projekt och därigenom minska tid och kostnader för inledande forskningsinsatser.

AVAINSANAT: Computer vision, object detection, deep learning, Convolutional Neural Networks, Automated guided vehicle

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Vähä, Eemil
Tutkielman nimi: An obstacle detection system for automated guided vehicles: [Subject]
Tutkinto:
Oppiaine: Energia ja informaatiotekniikka
Työn ohjaaja: Jouni Lampinen
Valmistumisvuosi: 2023 **Sivumäärä:** 112

TIIVISTELMÄ:
Tämän diplomityön tarkoituksena on tutkia sekä hyödyntää konenäköä ja hahmon tunnistusta osana erään automaattitrukin navigointijärjestelmää, joka oAperoi kohdeyritysken tiloissa. Motivaationa tämän tutkielman laatimiselle ja sovelluksen kehittämiselle on nykyisten automaattitrukkien huono kyky havainnoida pienempiä tai muuten epämääräisiä esineitä ja esteitä. Automaattitrukit eivät myöskään kykene erottamaan liikkuvia esineitä paikallaan pysyvistä, mikä aiheuttaa selkeän turvallisuusriskin sekä heikentää järjestelmän toimintakykyä. Tämän työn lopputuloksena odotetaan saavan konseptitodistus konenäkö applikaatiosta, joka voisi mahdollisesti parantaa trukin kykyä havainnoida esteitä sen kulkureitillä. Tarkoituksena on tuoda kohdeyritykselle käytäntöön perustuvaa tietoa sovelletusta konenäöstä kehittämällä YOLOv7 hahmontunnistusmalli ratkaisuna tutkimus ongelmalle.

Tämän tutkielman alku käsittää teoriaosan teknologioista ja tekniikoista, joita tällaisen hahmontunnistusmallin kehittämiseen vaaditaan, jonka jälkeen laaditaan suunnitelma sovellukselle vaatimusten määrittelyä varten. Koneoppimismallin kehittämiseen ja datan prosessointiin käytetään avoimen standardityökaluja sekä tietokirjastoja kuten Jupyter ja OpenCV. Ohjelmointikielenä käytetään pääasiassa Pythonia. Hahmontunnistusmallina käytetään YOLOv7 hahmontunnistus algoritmia, joka opetetaan havaitsemaan ja tunnistamaan määriteltyjä esteitä, jotka aiheuttavat ongelmia nykyiselle järjestelmälle. Mallin optimointi suoritetaan kokeilemalla eri määrityksiä, jonka jälkeen parhaan suorituskyvyn omaavaa mallia testataan simuloidussa ympäristössä. Data, jota tarvitaan hahmontunnistusmallin kehittämiseen, kerätään kiinnittämällä kamera automaattitrukkeihin ja kuvaamalla sen päivittäistä toimintaa kohdeyrityksen tiloissa. Datan annotointiin käytettiin Label Studio nimistä ohjelmistoa ja kaikki vaadittava datan valmistelu ja prosessointi suoritettiin käyttäen Pyhton ohjelmointi kieltä.

Tämän diplomityön tuloksena saatiin konseptitodistus konenäkösovellukselle, minkä käyttöönotto edistäisi kohdeyrityksen päivittäistä toimintaa nykyisissä tuotanto- ja varastointitiloissa. Kehitetyn mallin todettiin suoriutuvan riittävän hyvin odotuksiin nähden niin nopeuden kuin tarkkuuden osalta. Tämä projekti ei pelkästään todistanut sovelluksen hyötyä nykyiselle automaattitrukkijärjestelmälle, mutta loi myös perustan koneoppimisen laajemmalle hyödyntämiselle muualla liiketoiminnassa saadun tietotaidon myötä. Tämän diplomityön tulokset osoittivat, että optimaalisen objektintunnistusmallAin löytämine kohtuullisessa ajassa vaatii sekä asianmukaiset työkalut että riittävän tutkimustietopohjan mallin konfiguroinnin kannalta. Koulutettua mallia voitaisiin hyödyntää pohjana muissa vastaavissa hankkeissa ja siten vähentää esitutkimukseen kuluvaa aikaa sekä kustannuksia.

AVAINSANAT: Computer vision, object detection, deep learning, Convolutional Neural Networks, Automated guided vehicle

Table of contents

1 Introduction 12

1.1 Objective 14

1.2 Structure 15

1.3 Project plan 16

2 Computer vision 18

2.1 Object detection 18

2.2 Artificial intelligence 20

2.3 Deep learning 22

2.4 How deep learning works 23

2.4.1 Supervised and unsupervised learning 23

2.4.2 Reinforcement learning and hybrid learning 24

2.4.3 Nodes 25

2.4.4 Linear function 25

2.4.5 Activation function 26

2.4.6 Loss and cost function 27

2.4.7 Forward propagation 28

2.4.8 Backpropagation 28

2.4.9 Epoch 28

2.4.10 Gradient descent 29

2.4.11 Stochastic gradient descent (SGD) 30

2.5 Convolutional Neural Networks (CNN) 31

2.5.1 Input layer 32

2.5.2 Convolutional layer 33

2.5.3 Feature maps 34

2.5.4 Pooling layer 34

2.5.5 Fully connected layer 35

2.5.6 Output layer 36

2.6A Overfitting and underfitting 37
2.6.1 Learning rate and optimizer 38
2.6.2 Dropout 38
2.6.3 Regularization 39
2.6.4 Data augmentation 40
2.6.5 Error analysis 41
2.7 Data collection 41
2.7.1 Datasets 42
3 Project plan 44
3.1 System features and requirements 44
3.1.1 Premises and functional requirements 44
3.1.2 Non-functional requirements 50
3.1.3 External interface requirements 50
3.1.4 Quality attributes 51
4 Implementation 52
4.1 Yolov7 53
4.1.1 Architecture 53
4.1.2 Loss function 55
4.1.3 Metrics used with YOLOv7 56
4.1.4 Data augmentation 57
4.2 Data collection 58
4.3 Data preparation 58
4.4 Label studio 59
4.5 Model training 60
4.5.1 Training preferences 61
4.5.2 Model optimization 62
4.5.3 Test dataset 68
5 Testing 74
5.1 Test requirements 75
5.2 Test cases 76

5.3 Test results 80

6 Results and observations 84

7 Conclusions 89

References 92

Appendices

Appendix 1. List of objects AGVs encounter when operating 1

Appendix 2. Python commands 1

Appendix 3. Hyperparameters of the tenth model 1

Appendix 4. Training results 1

Appendix 5. Test cases 1

Appendix 6. Dependencies 1

Pictures

Picture 1 Communication between the EWM system, AGV PC, and the AGVs 12
Picture 2 Visualization of the AGVs safety scanners and navigation system 13
Picture 3 Figure illustrating basic principles of object detection. (Dadhich, 2018)..... 19
Picture 4 Relations between the term's artificial intelligence, machine learning, and deep learning (IBM, 2022) 21
Picture 5 General structure of a deep neural network (IBM, 2022) 24
Picture 6 An overview of a node in a neural network (Chokmani;Khalil;Ouarda;& Bourdages, 2007)..... 25
Picture 7 Gradient decent (Haji & Abdulazeez, 2021) 29
Picture 8 Characteristics of a cost function (Goodfellow;Bengio;& Courville, 2018).... 30
Picture 9 Architecture of LeNet-5 a Convolutional Neural Network (LeCun;Bottou;Bengio;& Haffner, 1998)..... 32
Picture 10 Convolution between an image data matrix and kernel 33
Picture 11 Max pooling..... 35
Picture 12 A visualization of a fully connected layer (Kost;Altabey;Noori;& Taher, 2019) 36
Picture 13 Graphs visualizing under fit, balanced, and overfitting of a machine learning model (Amazon, 2023) 37
Picture 14 A neural net before and after applying dropout (Srivastava;Hinton;Krizhevsky;Sutskever;& Salakhutdinov, 2014) 39
Picture 15 Data augmentation achieved with Geometric deformation (Yorioka;Kang;& Iwamura, 2020)..... 40
Picture 16 Example of a training batch with mosaic data augmentation used to train the model..... 57
Picture 17 Label Studio user interface..... 60
Picture 18 Example of a linear motion labelling sequence in Label Studio 60
Picture 19 The horizontal black solid line represents the limit that an object cannot surpass, and the vertical dashed lines illustrates the width of the AGV. A pallet-jack is classified with a confidence of 95%..... 76

Picture 20 Test cases for pallet-jack. 77
Picture 21 Test cases for rider-truck. 78
Picture 22 Test cases for reach-truck. 79
Picture 23 Test cases for counterbalance-truck 80
Picture 24 Example of a training batch used to train model number ten. 6
Picture 25 Test case 1 for pallet-jack. 1
Picture 26 Test case 2 for pallet-jack. 1
Picture 27 Test case 1 for rider-truck. 2
Picture 28 Test-case 2 for rider-truck 2
Picture 29 Test case 1 for reach-truck. 3
Picture 30 Test case 2 for reach-truck. 3
Picture 31 Test case 1 for counterweight-truck. 4
Picture 32 Test case 2 for counterweight-truck. 4

Figures

Figure 1 Project plan in form of a Gantt chart 17
Figure 2 Process flowchart to determine detectable objects 46
Figure 3 Braking distance-speed graph for soft stop 47
Figure 4 Braking distance-speed graph for emergency stop 48
Figure 5 Development process 52
Figure 6 Yolov3 architecture with introduction to PP-Yolo features (Xiang, ym., 2020) 54
Figure 7 Extended efficient layer aggregation network (Wang;Bochkovskiy;& Mark Liao, 2022) 54
Figure 8 Results of second training run. 64
Figure 9 F1 curve after the fourth training run. 65
Figure 10 Confusion matrix of training run six. 65
Figure 11 Results of the eighth training run with learning rate and lower mosaic parameter. 66

Figure 12 The tenth model's performance on validation and test dataset and average performance of all models. 69
Figure 13 Distance estimation between an object and the AGV. 76
Figure 14 Number of false detections in frames by confidence threshold and object class. 83
Figure 15 Results of the third training run. 1
Figure 16 Results of the fifth training run with a larger dataset. 1
Figure 17 F1 score of the fifth training run. 2
Figure 18 Results of seventh training run with lower mosaic hyperparameter. 2
Figure 19 Confusion matrix after the seventh training run. 3
Figure 20 Results of the ninth training run with lower learning rate and initial mosaic parameter. 3
Figure 21 Confusion matrix of the ninth training run. 4
Figure 22 Results of the tenth training run with adjusted augmentation parameters. ... 4
Figure 23 Confusion matrix of the tenth training run. 5

Tables

Table 1 Popular Activation functions (Pragati, 2023) 27
Table 2 List of objects that the model is trained to detect and classify. 46
Table 3 First dataset used for training. 61
Table 4 Second dataset used for training. 62
Table 5 Final dataset used for training. 62
Table 6 All training runs with respective parameters. 63
Table 7 Overall performance of each model. 67
Table 8 Test results with the highest scores highlighted with respect to the metric. 71
Table 9 Summary table of the trained model 73
Table 10 Test results by class and evaluation criteria. 81
Table 11 Number of frames it took for the model to detect and classify the objects by test case. 81
Table 12 Results for testing number of false detection in frames. 82

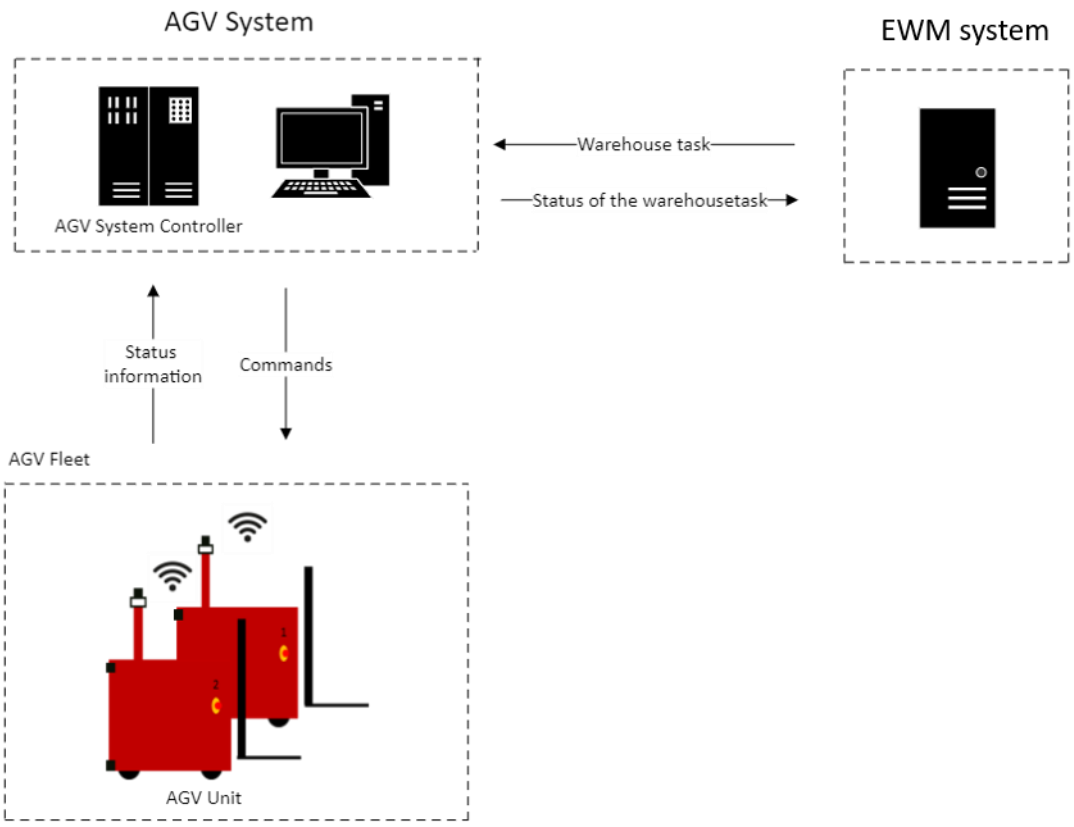
Table 13 False detections by duration. 82

Abbreviations

AGV	Automated guided vehicle
ANN	Artificial neural network
AP	Average Precision
CNN	Convolutional neural network
ERP	Enterprise resource planning system
EWM	Extended warehouse management
FN	False Negative
FP	False Positive
FPS	Frames Per Second
IoU	Intersect over Union
mAP	mean Average Precision
OIC	Operational improvement competency
STH	Sustainable technology hub
TP	True Positive
YOLO	You Only Look Once, an object detection algorithm named YOLO

1 Introduction

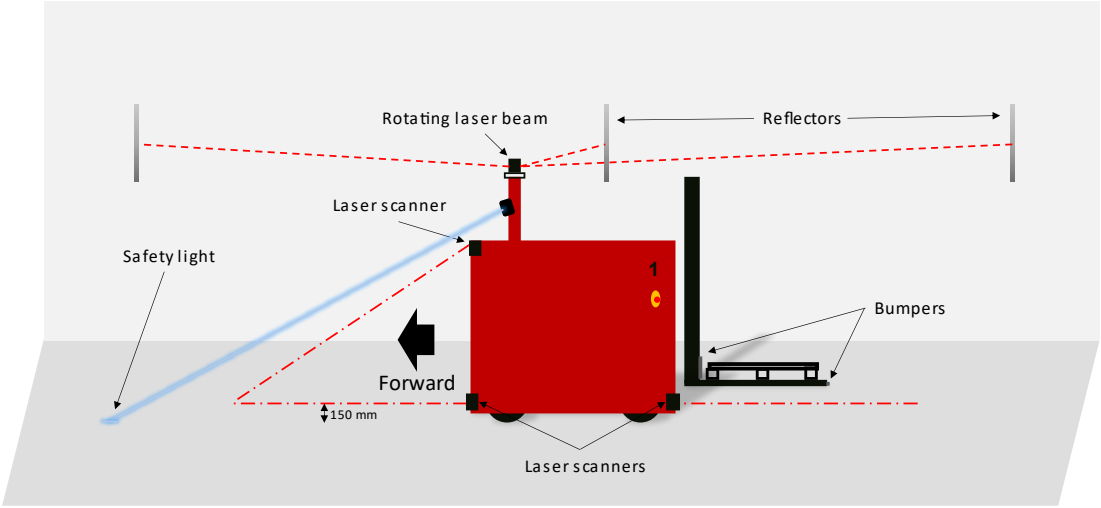
In 2022, the construction phase of Wärtsilä's forthcoming logistics centre and factory in Vaasa reached completion, commencing the implementation phase of a new logistical operations model. This model comprises a new extended warehouse management system (EWM) and an integrated automated guided vehicle (AGV) system. The fundamental concept underlying this model is that the EWM system assigns tasks to the AGVs, which subsequently execute these tasks as instructed. The AGV fleet consists of six automated vehicles, which all perform identical tasks being namely transportation of euro pallets from location A to location B. These tasks, commonly known as "warehouse tasks", entail specific instructions for the pickup and delivery of euro pallets.



Picture 1 Communication between the EWM system, AGV PC, and the AGVs

The AGVs use laser navigation to navigate the warehouse and factory. A real-time route calculation is done based on the locations of reflectors installed in the building that a

spinning laser beam on top of the AGV detects (Solving, 2022). This means that the AGVs always use a predetermined fixed path and do not deviate from it independently. The automated guided vehicles (AGVs) are equipped with laser scanners and bumpers to detect obstacles. The AGVs feature three laser scanners located in the front and one in the back. One of the laser scanners is positioned top of the AGV and is angled 45 degrees towards the ground. Meanwhile, the remaining two front scanners and the one at the back are located approximately 100mm from the ground level on the AGV's lower section. These scanners function in parallel with the ground, thereby creating a two-dimensional plane encompassing the AGV.



Picture 2 Visualization of the AGVs safety scanners and navigation system

If the laser scanners detect an obstacle, it slows down and stops if the obstacle is not removed. However, the AGVs cannot detect objects under 100 mm high and lying on the floor. Neither are they able to detect obstacles horizontally in the air, e.g., spikes of a forklift. This does, of course, cause a safety risk and is currently avoidable only by precautionary measures. In practice, this would mean that somebody would have to make sure that the AGV paths are clear of all possible undetectable objects all the time. This is different from the planned course of action since the presumption is that the AGVs would be able to operate independently without supervision.

1.1 Objective

The purpose of this master thesis is to create a proof of concept for a computer vision application by training an object detection that could be utilized to improve the current AGV system's performance in terms of avoiding crashing into obstacles. The general idea is that a camera is placed on top of an AGV, facing the direction AGVs move when operating. Video data from the camera would then be fed to an embedded system capable of running the required object detection model. The embedded system would analyse the data and signal the AGV to stop if an object is detected in its path based on a predefined set of criteria. This master's thesis is conducted as an assignment to Wärtsilä in return for a compensation that is paid as a lump-sum after the thesis is finalized.

All possible variables must be considered in order to successfully train an object detection model that meets all the requirements that an application of this nature sets. The nature of the dataset used to train the model depends on not only the observed objects but also noise levels, footage quality, and lighting intensity. The number of ways a computer vision application could be developed considering the number of available open-source object detectors and data processing techniques, is quite exhaustive. However, most of them might not give the desired output because they do not fit the nature of the dataset in question. Therefore, preliminary research on the topic is fundamental (Davies, 2017).

Nowadays, computer vision is utilized widely in different applications. Recent developments in deep learning techniques have increased the accuracy of applications like object detection, tracking, and image classification, which has triggered the development of more complicated autonomous systems like drones, self-driving automobiles, humanoids, and other things (Dadhich, 2018). In addition, the constantly growing number of computer vision applications, research papers and books written

about computer vision applications, and available open-source datasets all s the work required to conduct this master's thesis.

To showcase an example of valuable references considering this master's thesis, the Technical University of Munich claims that they have put together the first annotated dataset with a logistics-related focus available to the public. This data set consists of over 39 thousand images of logistics-specific objects such as pallets, small load carriers, stillages, forklifts, and pallet trucks which are common objects within the logistics domain (Mayershofer;Holm;Molter;& Fottner, 2022). This dataset could be, e.g., utilized to pre-train or test an object detection model. In addition, there are over one thousand search results on conference papers and books in IEEE Xplore digital library with the entry "Computer vision" to be included in the document title within the last five years.

The main objective of this thesis is to provide practical knowledge about the usability of computer vision in Wärtsilä's business operations in form of a proof of concept for a computer vision application. The gained knowledge helps further develop the application and possibly utilize the same technology in other business areas. However, there is a lot to improve considering both the safety and efficiency of the automated guided vehicle system, and that is what the outcome of this master's thesis should primarily improve. This master's thesis aims to answer the following research question. "Can object detection be utilized to improve the current automated guided vehicle system's performance in terms of avoiding crashing into obstacles it cannot detect with its current scanners?"

1.2 Structure

This thesis can be divided into four main sections. First, the introduction section containing chapter 1 defines the current problem narrowing down the scope of this thesis into a specific research question that the study seeks to answer. The following section is the methods section containing chapters 2 and 3, where all relevant theory

related to the topic of this master’s thesis is explained. This helps the reader to better understand the methods used to carry out the training of an object detection model.

The primary body of the research comprises the methods and the results section, which comprises of rest of the chapters. The implementation section containing chapters 4 and 5 outlines the actual development carried out during the research, which is then discussed in the last section of this thesis. Finally, the discussion section including chapters 6 and 7, discuss the study’s outcome and present its impact on the current problem and the broader implications of the research.

1.3 Project plan

To provide a clear and comprehensive overview of the timeframe for conducting this master's thesis, a Gantt chart has been utilized and is presented in Figure 1. The Gantt chart displays all significant phases involved in the process of conducting this master's thesis. This chart serves as a visual aid, helping to illustrate the entire duration of the project and enabling stakeholders to understand the various stages involved. By using the Gantt chart, the timeline for conducting this master's thesis can be effectively monitored, ensuring that the project remains on track and within the designated time frame.

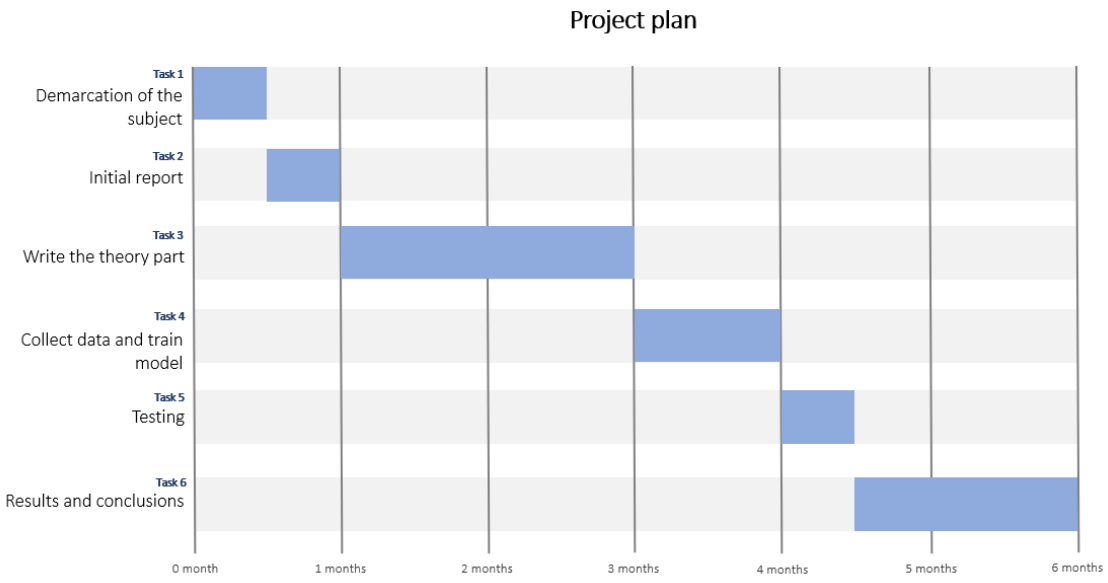


Figure 1 Project plan in form of a Gantt chart

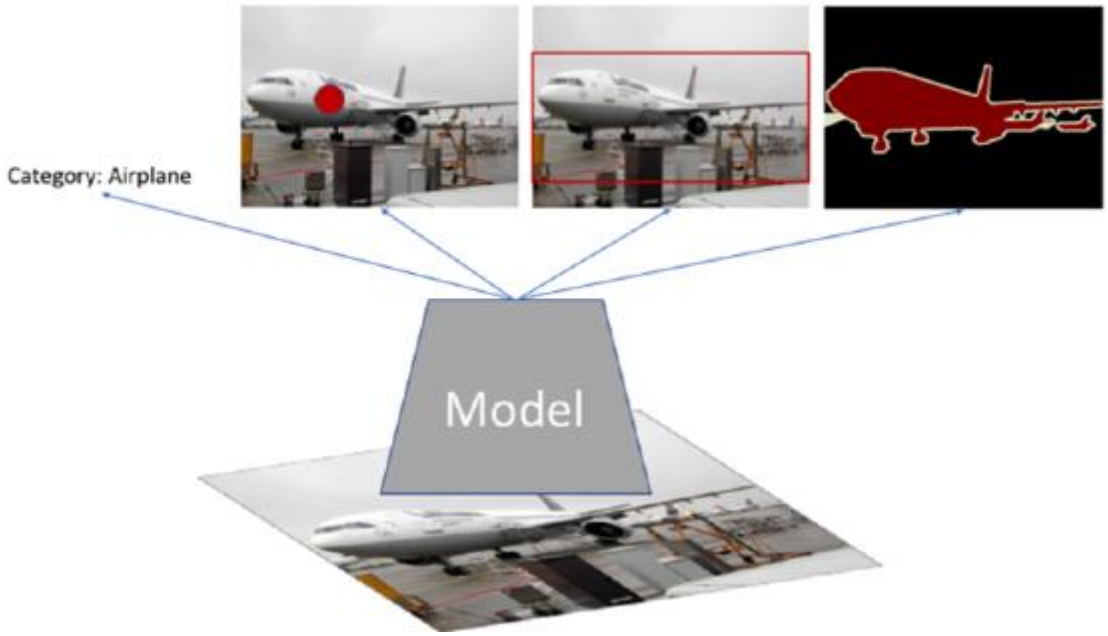
2 Computer vision

For the reader to properly understand the general concepts and methods related to computer vision and object detection in general, a sufficient review of the topic is fundamental. Computer vision is a broad term for methods and underlying theories related to image processing. The definition of what distinguishes computer vision from image processing, pattern recognition, or machine vision varies, and the boundary between them needs to be clarified (Fisher, ym., 2014). In the book “Dictionary of Computer Vision and Image Processing” (2014), there are over three and a half thousand terms related to computer vision and image processing. However, this does not cover the most recent ones related to the latest developments in deep learning techniques. However, this chapter only explains terms, methods, techniques, and the related theory affiliated with this master’s thesis.

The underlying technology is usually computer vision when discussing practical applications capable of detecting and identifying objects from image and video data. Computer vision can be utilized in various applications like image classification, object detection, object tracking, image geometry, image segmentation, and image generation (Dadhich, 2018). The primary emphasis of this master's thesis will be on object detection, as well as related techniques and tools.

2.1 Object detection

In short terms, object detection is about localizing objects in images and identifying what the object is based on predefined specifications. A generic image recognition model where an image is passed through the model, which gives information as output about the class name for the object in the image, the object centre pixel, a bounding box surrounding the object, and the class for each pixel in the image. The first and third are usually affiliated with computer vision (Dadhich, 2018).



Picture 3 Figure illustrating basic principles of object detection. **(Dadhich, 2018)**

Achieving vision artificially is a challenge on its own. The problem is complex because it is an inverse problem, where the solution is based on inadequate data to specify the solution fully. In computer vision, this problem is tackled by seeking a solution by applying machine learning to clarify the differences between possible solutions, i.e., trying to interpret the visual world with images and reconstructing properties, such as shape, illumination, and colour distributions (Szeliski, 2022).

Modelling the real world is nearly impossible, considering all the possible challenges encountered when developing a computer vision application in the ever-changing, complex real world. Common challenges that need to be acknowledged when working with an object detection application are occlusion, viewpoint changes, variation in size, non-rigid objects, and motion blur, especially in an environment where these change a lot (Dadhich, 2018). These are all discussed in more detail in chapter 4 when the actual model is trained.

Object detection is just one computer vision technique that helps, e.g., detect objects in an image. However, there are many different variations of object detection models

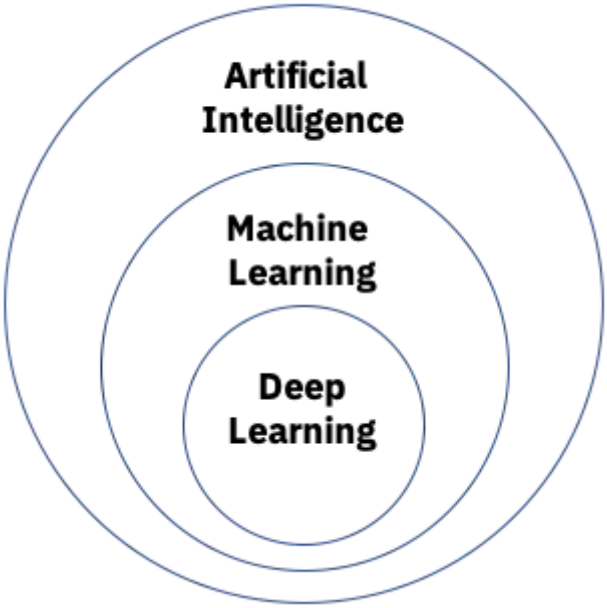
having different types of architectures that suit certain types of applications. In order to get the best-fitting and performing network for a specific application, comprehensive research on the topic is fundamental (Zhong-Qiu;Peng;Shou-Tao;& Xindong, 209). The object detection algorithm used to train the model in this master's thesis is YOLOv7. A more detailed explanation of YOLOv7 is laid out in chapter 4.1.

2.2 Artificial intelligence

The term artificial intelligence is somewhat elusive. However, it is beneficial to understand the relations between the different concepts to get a clear overview and understanding of the technologies used in this thesis. One of the founding fathers of the term "artificial intelligence" John McCarthy (2007, s. 2) defines it the following way in his paper "What is artificial intelligence?":

> *It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.*

Artificial intelligence is a field that, in its simplest basic form, combines computer science and substantial datasets to facilitate problem-solving. Additionally, it includes the subfields of artificial intelligence known as machine learning and deep learning, which are commonly addressed together. These fields use AI algorithms to build intelligent systems that predict or categorize information based on incoming data (IBM, 2022).



Picture 4 Relations between the term's artificial intelligence, machine learning, and deep learning **(IBM, 2022)**

Machine learning and deep learning are both subfields of artificial intelligence. The key difference between machine learning and deep learning is in the method how learning is accomplished. Machine learning is essentially artificial intelligence that is more dependent on human intervention when it comes to learning. It makes more linear and simple correlations and is usually trained on smaller datasets. Whereas deep learning is a subfield of machine learning that relies on supervised learning with artificial neural networks (IBM, 2022). Deep learning essentially a mathematically more complex evolution of traditional machine learning algorithms. They both use same learning techniques being both supervised and unsupervised but deep learning utilize more complex layered structure of algorithms known as artificial neural networks. A typical deep learning architecture is a convolutional neural network that consists of a fully connected neural network that is capable of learning complex non-linear information from large datasets and thereby prospering in the field of image classification and is therefore useful in computer vision (Campesato, 2020).

In general, deep learning models outperform traditional machine learning methods when the learning dataset grows and is therefore used in applications with large amounts of data. It is to be noted that deep learning requires much more data than traditional machine learning due to the complex multilayer structure deep learning utilizes in order to make accurate inference (Amitha;Amudha;& Sivakumari, 2020).

2.3 Deep learning

Deep learning or deep models, also referred to as neural networks, are a core element of computer vision-based object detection. Neural networks date back to the 1940s when they were used to simulate the human brain system. The popularity of neural networks increased in the 1980s and the 1990s when a back-propagation algorithm was proposed in a journal article but eventually died due to a lack of computational resources (Zhong-Qiu;Peng;Shou-Tao;& Xindong, 209).

Deep learning comprises different types of neural network architectures such as convolutional neural network (CNN), recurrent neural network (RNN), long short-term memory (LSTM), and gated recurrent unit (GRU) (Campesato, 2020). Convolutional neural networks are the most relevant of these architectures, considering the topic of this master's thesis. From a more general high-level perspective, deep learning with supervised learning generally consists of defining the neural network, estimating data point, i.e., labelling the dataset, calculating the loss or error of each estimate, and reducing the error via gradient descent (Campesato, 2020). These steps are explained in this chapter including a general overview of CNN architecture.

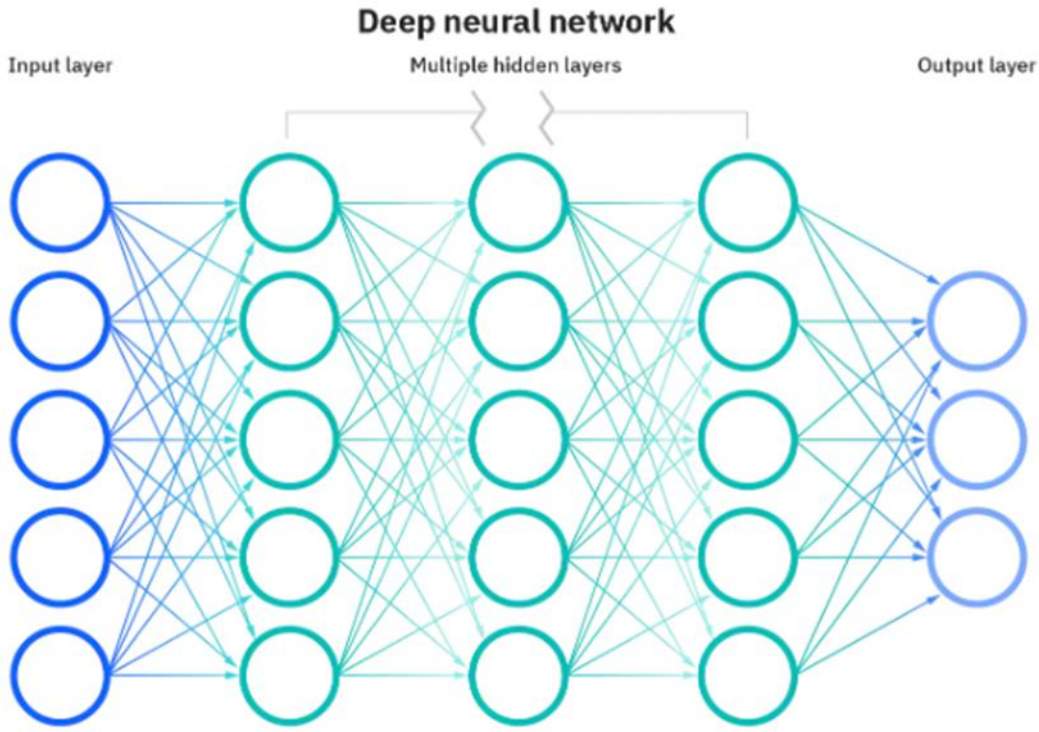
In addition to the well-established deep learning architectures, there exist several relatively recent advancements, including input convex neural networks (ICNN) (Amos;Xu;& Kolter, 2017) and global optimization networks (GON) developed by (Zhao;Louidor;& Gupta, 2022). These architectures represent cutting-edge developments in the field of deep learning and have shown significant promise in improving the performance and accuracy of object detection systems.

2.4 How deep learning works

The general structure of a deep learning neural network requires at least two hidden layers whereas very deep learning would require at least ten hidden layers. These layers consist of connected nodes or often referred to as neurons, each of which builds on the one before it to improve and refine the prediction or classification. The number of hidden layers can be adjusted in the training process with a parameter also known as a hyperparameter (Davies, 2017). In general, deep neural networks are trained with supervised learning, unsupervised learning, reinforcement learning, as well as hybrid learning (Amitha;Amudha;& Sivakumari, 2020).

2.4.1 Supervised and unsupervised learning

Supervised and unsupervised learning are both machine learning techniques. In supervised learning, the task is to predict an output y given an input x based on a training dataset. I.e., supervised learning uses labelled datasets to train the machine learning model or neural network. Unsupervised learning relies on detecting and identifying patterns or structures in data with algorithms without human interaction (Fisher, et al., 2014). Generally deep learning relies on supervised learning in object detection and classification-related tasks (Goodfellow, Bengio, & Courville, 2018).



Picture 5 General structure of a deep neural network (IBM, 2022)

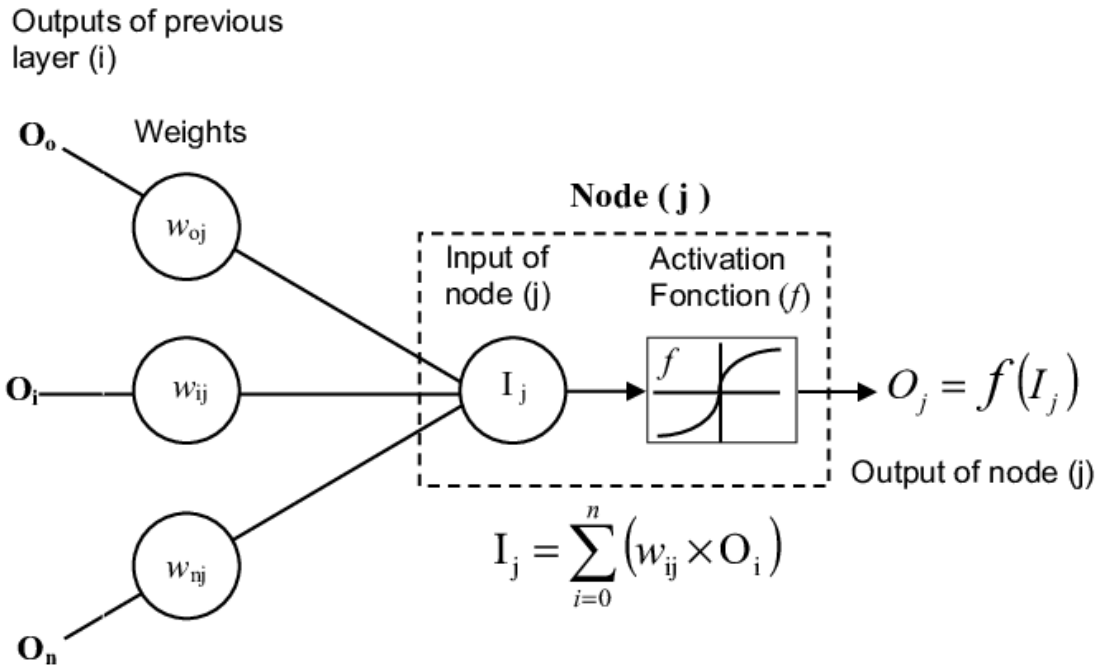
2.4.2 Reinforcement learning and hybrid learning

Reinforcement learning is a machine learning method that is based on rewarding actions that have a positive impact and punishing actions that have a negative impact on the environment. Essentially the goal is to a maximize the total reward of the agent making the actions based on the state of the environment. The agent learns in an interactive environment by trial and error with the help of the feedback from its own actions (Amitha;Amudha;& Sivakumari, 2020).

Hybrid learning on the other hand is a deep neural network that is a combination different architectures and utilize generative (unsupervised) as well as discriminative (supervised) components to learn. In general, generative models can generate new data instances whereas discriminative models only discriminate between different kinds of data inputs (Kuleshov & Ermon, 2017).

2.4.3 Nodes

A node is the smallest building element of a neural network and is responsible for adjusting the variable values processed by the network. Most of the nodes or neurons in the network consist of two different functions. The nodes are composed of linear and activation functions (Bernico, 2018). The number of neurons is also adjustable with a hyperparameters and is generally part of the initial set-up phase of a neural network (Campesato, 2020).



Picture 6 An overview of a node in a neural network **(Chokmani;Khalil;Ouarda;& Bourdages, 2007)**

2.4.4 Linear function

The function in a linear function is an essentially linear regression function that calculates a weighted sum based on the inputs by multiplying each input with a coefficient, also known as a weight. The final output z of a linear function is calculated with function 1 where $\{x_1, x_2, \ldots, x_n\}$ are the inputs and $\{w_1, w_2, \ldots, w_n\}$ are the weights

or coefficients. The values of the weighted sums are calculated in each neuron in a layer where after the same process is repeated in the next layer (Bernico, 2018).

$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n \quad (1)$$

2.4.5 Activation function

After calculating the weighted sum with a linear function, the value is fed to the following function called an activation function. An activation function is a non-linear function used in neural networks to introduce non-linearity to the model. A neural network without an activation function would essentially be a linear regression model performing linear transformation to the inputs of the neurons (Bernico, 2018).

A few examples of common activation functions used in machine learning and deep learning are Sigmoid, TanH, and ReLU, each with properties suitable for different types of problems. The difference between these activation functions is the function that produces a specific output range for each function (Pomerat;Segev;& Datta, 2019).

Activation function	**Function**	**Output range**
Sigmoid	$sigmoid(x) = \frac{1}{1+e^{-x}}$	[0,1]
TanH	$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	[-1,1]
ReLU	$f(x) = \max(0, x)$	[0, ∞]
Binary Step	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	[0,1]
Linear	$f(x) = x$	[-∞, ∞]
Leaky ReLU	$f(x) = \max(0.1 * x, x)$	[0, ∞]
Parametric ReLU	$f(x) = \max(ax, x)$	[0, ∞]

ELU	$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$	[-1, ∞]
Softmax	$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$	[0,1]
Swish	$f(x) = x * sigmoid(x)$	[0,1]
GELU	$f(x) = 0.5x(1 + tanh\sqrt{\frac{2}{\pi}}(x + 0.044715x^3))$	[-1, ∞]
SELU	$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$	[-1, ∞]

Table 1 Popular Activation functions **(Pragati, 2023)**

2.4.6 Loss and cost function

A loss function measures how well a neural network predicts the correct output for a given input. The loss function reduces the difference between an expected and actual output so that the network can learn to predict more accurately. At the same time, a cost function is a measure of the network's overall error and is calculated by averaging the loss function across all training examples. The goal is to find the weights and biases that minimize the cost function, which is achieved with an optimizer in deep learning. A model is said to converge when the loss settles within an error range around the final value, i.e., when training does not improve the model's performance anymore (Yingjie;Duo;Stanislao;& Xiaohui, 2022).

Cross entropy-based loss function is one of the most common types of loss function used in deep neural networks, especially in classification related tasks such as object detection. Cross-entropy loss is calculated with equation 2 where g_j is the discrete ground-truth label of class j, and p_j *is* the output of the output layer of the network also known as the softmax layer (Yingjie;Duo;Stanislao;& Xiaohui, 2022).

$$L_{CE} = -\sum_j g_j \log(p_j) \quad (2)$$

2.4.7 Forward propagation

Forward propagation is the process in deep learning where the input data is processed within the neural network, starting from the input layer, and ending at the output layer, eventually passing multiple hidden layers if present. During forward propagation, the input data is transformed through a series of computations, also known as "forward computations," in each layer of the network until it reaches the output layer, where the result or prediction is produced. The outputs are based on the weights of the linear functions the neurons in each layer, as well as the activation function. The forward propagation process is used to predict the desired target variable based on the input data and the parameters of the trained neural network (Bernico, 2018).

2.4.8 Backpropagation

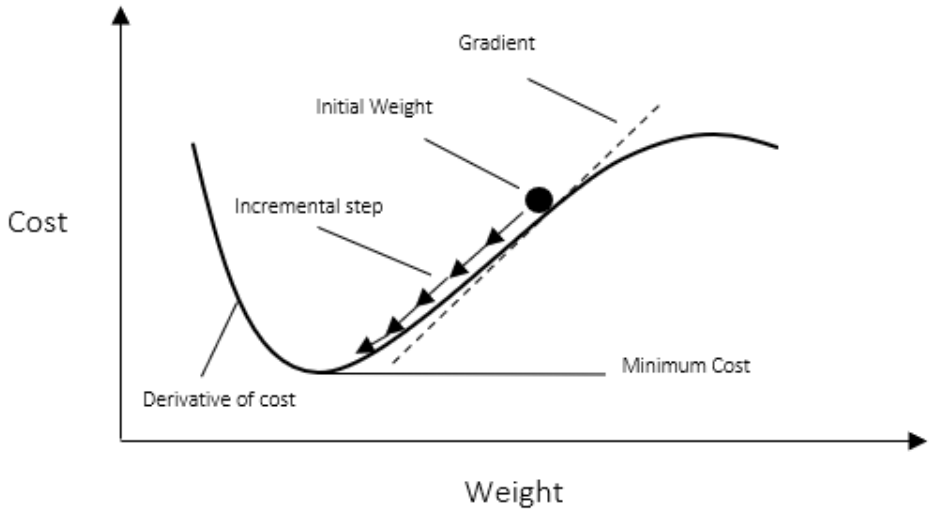
Deep neural networks gained popularity in the 1980s and 1990s when the concept of backpropagation was presented by Rumelhart et al. in a nature article (Rumelhart;Hinton;& Williams, 1986). The actual training of the neural network happens when updating the weights of the neurons based on the network's error calculated with the cost function. The goal of backpropagation is to adjust the weights of the neurons in the neural network so that the network would make a more accurate prediction. This is accomplished with a backpropagation process that calculates the gradient using the chain rule of calculus. However, backpropagation refers only to the operation of calculating the gradient, which is then used to train the network with an optimizer, such as stochastic gradient descent (Goodfellow;Bengio;& Courville, 2018).

2.4.9 Epoch

Each round in the training process is called an epoch. During each epoch, the entire network dataset is processed, meaning that the model's parameters, such as weights, are updated based on the backpropagation process. The number of epochs eventually determines how long a model is trained and it can be adjusted with a hyperparameter during the training process (Campesato, 2020).

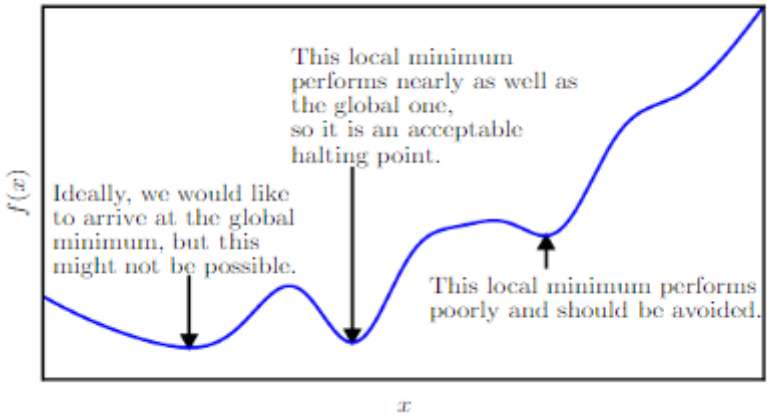
2.4.10 Gradient descent

Deep learning usually introduces some form of optimization. Optimization refers to the task of either minimizing or maximizing some function by altering some variable. In this case, the cost function is the function to be minimized, and the weights of the nodes in the network are the variables altered (Goodfellow;Bengio;& Courville, 2018). In deep learning, an optimizer is an algorithm used to minimize the loss function, and the two most common optimizers are Gradient Descent and Stochastic Gradient Descent (Pomerat;Segev;& Datta, 2019).



Picture 7 Gradient decent **(Haji & Abdulazeez, 2021)**

The principle of gradient decent is to take repeated steps in the opposite direction of the gradient on a specific point of the function cost function. This is considered the steepest descent direction of the function. However, taking a step in the other direction results in a local maximum referred to as gradient ascent (Goodfellow;Bengio;& Courville, 2018).



Picture 8 Characteristics of a cost function **(Goodfellow;Bengio;& Courville, 2018)**

The size of these steps is called the learning rate (α). With a large learning rate, the algorithm covers more ground in each step but has a chance of overshooting and completely missing the desired global minimum. With a low learning rate, this risk is neglectable; however, being a more time-consuming process and has the risk of being stuck in the local minima of the function. The learning rate can also be adjusted with a hyperparameter during training and is often used to stabilize the process. The most common learnings used in deep learning are 0.001, 0.003, 0.01, 0.03, 0.1, and 0.3 (Haji & Abdulazeez, 2021).

2.4.11 Stochastic gradient descent (SGD)

One of the most common optimizers or learning methods in deep learning and machine learning is stochastic gradient descent (SGD), also known as the online update. Stochastic gradient descent converges faster than regular gradient descent (LeCun;Bottou;Bengio;& Haffner, 1998).

In short, gradient descent is an iterative algorithm that descends a function's slope in steps from a random point until it reaches its lowest point. The learning rate is an important parameter that has an impact on the convergence of the algorithm since it determines the size of each step and is usually chosen by trial and error rather than

based on any general guidance (Goodfellow;Bengio;& Courville, 2018). Stochastic gradient modifies the network's configuration after each training point in an effort to locate the global minimum. Instead of reducing the error or determining the gradient for the complete data set, the gradient is calculated based on a randomly chosen batch of samples, which could be as small as a single sample. In practice, this is achieved by randomly shuffling the dataset and moving through the batches in steps, making it considerably faster and computationally more efficient than regular gradient descent. SGD is, therefore, suitable for large-scale datasets (Bottou, 2018).

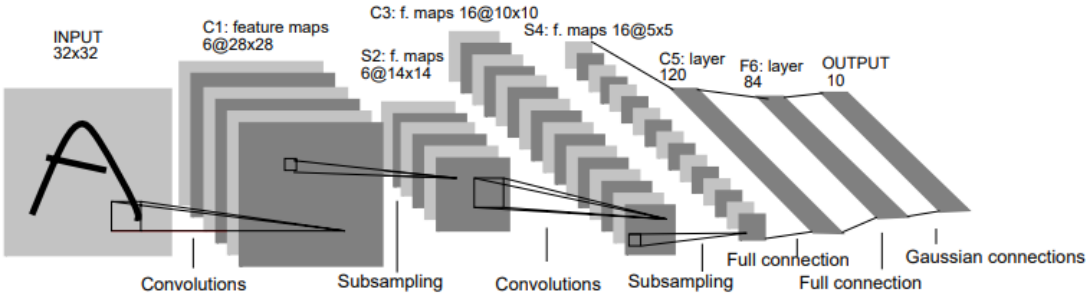
Other optimizers used to optimize neural networks in deep learning are Momentum, Nestrov Momentum, Adaptive Gradient Descent (AdaGrad), Adaptive Delta (AdaDelta), Root Mean Square Propagation (RMSProp), Adaptive Moment Estimation (Adam), and Maximum Adaptive Moment Estimation (AdaMax) each having its distinguishing features (Haji & Abdulazeez, 2021). However, explaining all of them would be irrelevant. The critical takeaway is to understand that optimization algorithms are the backbone of the learning process in deep learning neural networks (Goodfellow;Bengio;& Courville, 2018).

Other widely used deep learning methods are learning rate decay, dropout max-pooling, batch normalization, skip-gram, and transfer learning each used to and optimizing the model and solve different problems as well as reduce training time. E.g., dropout which is also explained in more detail in chapter 2.6 as a method to address overfitting (Amitha;Amudha;& Sivakumari, 2020).

2.5 Convolutional Neural Networks (CNN)

Recent computer vision advances have made deep learning a noteworthy and advantageous asset consumers utilize more often, especially in the commercial sector where convolutional neural networks were used to solve critical commercial applications such as AT&T optical character recognition (OCR) application for reading checks in 1998 and remaining the optimum solution up to today (Goodfellow;Bengio;& Courville, 2018).

Convolutional neural networks are a very versatile and relatively straightforward model in theoretical terms, however, being highly applicable to various perceptual tasks such as object detection. Convolutional neural networks are a trainable neural network architecture capable of learning invariable features from large sets of labelled data using an architecture composed of stages consisting of a filter bank layer, a non-linearity layer, and a feature pooling layer (LeCun;Kavukcuoglu;& Farabet, Convolutional Networks and Applications in Vision, 2010).



Picture 9 Architecture of LeNet-5 a Convolutional Neural Network **(LeCun;Bottou;Bengio;& Haffner, 1998)**

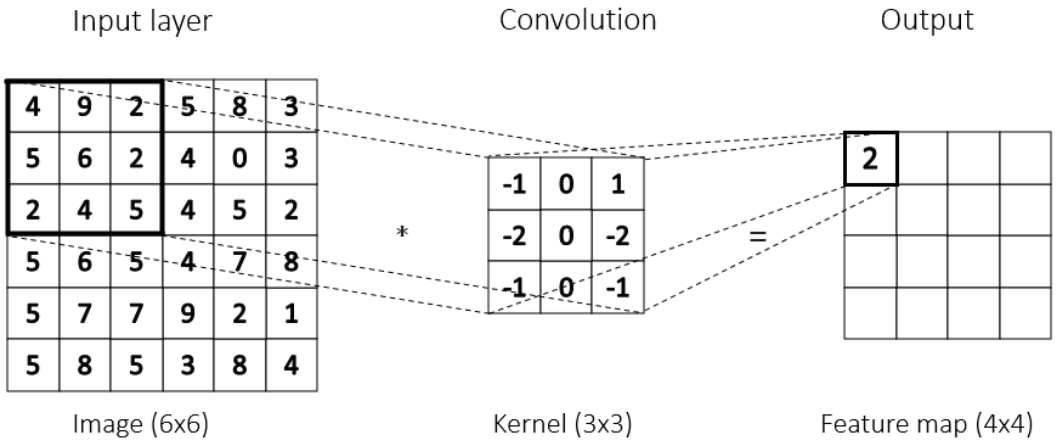
This chapter explains all the relevant terms and concepts related to a convolutional neural network so that the reader would have a general idea of how convolutional neural networks work and how they are utilized in this master's thesis.

2.5.1 Input layer

Convolutional neural networks are neural networks specialized in processing data with a grid-like topology, such as image data. The input layer is the first layer and the input of the whole convolutional neural network. In the context of a neural network applied to image processing, the input data is a pixel matrix of an image (Zhang ;Wang;Zhang ;Xu ;& Chen, 2019).

2.5.2 Convolutional layer

Instead of general matrix multiplication, convolutional neural networks employ a mathematical operation called convolution in at least one layer of the architecture (Goodfellow;Bengio;& Courville, 2018). In the context of neural networks, convolution can be considered a mathematical operation that combines two sequences or sets of numbers to produce a third sequence. A 2D image's convolution operation is defined as the weighted sum of the components in a small window, known as a kernel or filter, moving over the image. These weights are learned parameters modified in the kernel during training to detect features in the image such as edges, lines, and corners (Zhang ;Wang;Zhang ;Xu ;& Chen, 2019). This is employed in the convolutional layer, the second layer of a CNN architecture. The output of the convolutional layer is often referred to as a feature map containing real numbers (Campesato, 2020).



Picture 10 Convolution between an image data matrix and kernel

Depending on the nature of the computer vision application, image processing can improve the model's performance significantly. Image processing is therefore a crucial part of training a deep learning model for a computer vision application. This is achieved by applying filters or kernels to the original image. The purpose of these filters is to process the image, e.g., remove noise, blur an image, extract edges, remove objects or

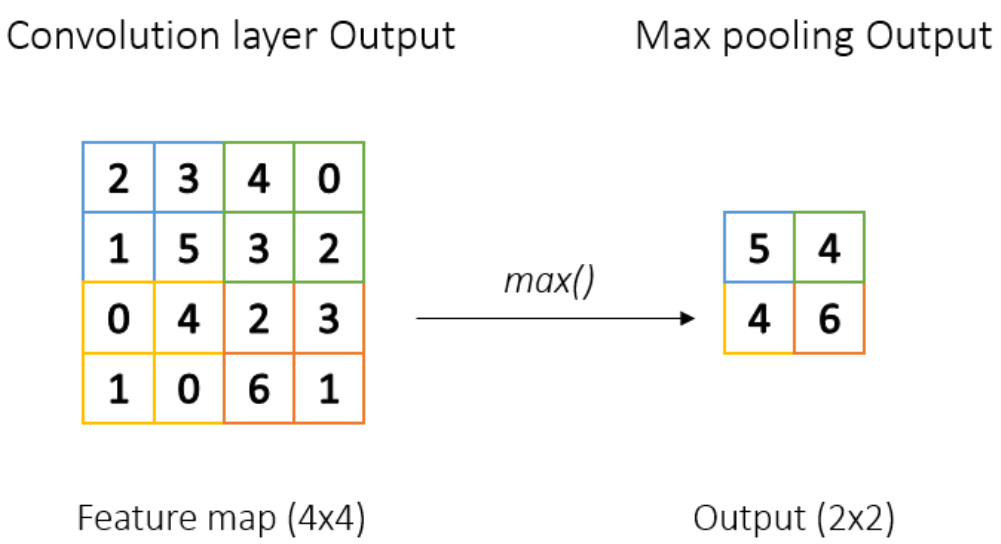
highlight objects (Nixon & Aguado, 2020). For example, the kernel in picture 10 is called a Sobel, usually used to highlight edges in an image (Sanida;Sideris;& Dasygenis, 2020).

2.5.3 Feature maps

During the second stage of the object detection process, feature maps generated by the convolutional layer are processed through non-linear activation functions, typically a Rectified Linear Unit (ReLU) activation function. The ReLU activation function sets negative values in the feature maps to zero, introducing non-linearity in the network. This helps the network identify and highlight important object features, ultimately improving the precision and effectiveness of the object detection system (Campesato, 2020).

2.5.4 Pooling layer

The third stage of convolutional neural networks is the pooling layer which has the role to merge semantically similar features from the output of the previous layer into one (LeCun;Bengio;& Geoffrey, Deep Learning, 2015). This is achieved with a pooling function that replaces the output of a feature map with a statistical summary of the neighbouring output values. There are several pooling functions used in convolutional neural networks such as max pooling, average of a rectangular neighbourhood, the L^2 norm of a rectangular neighbourhood, or a weighted average based on the distance from the central pixel (Goodfellow;Bengio;& Courville, 2018).

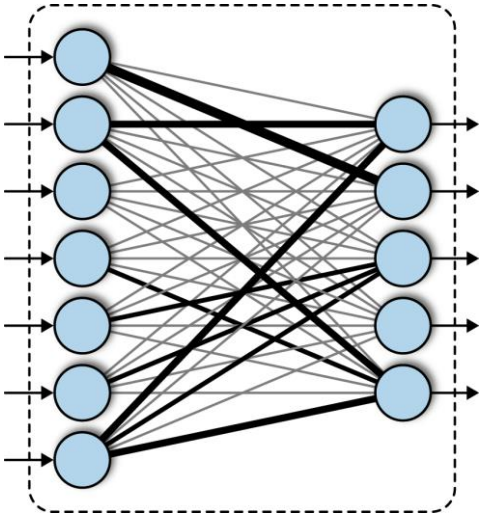


Picture 11 Max pooling

A typical pooling function is max pooling which is performed by applying a max filter to non-overlapping sub-regions of the original output i.e., taking the maximum value of these regions. Applying any kind of pooling to a makes the pooled output invariant to smaller changes in the input image (LeCun;Kavukcuoglu;& Farabet, 2010).

2.5.5 Fully connected layer

The fourth layer of a convolutional neural network is the fully connected layer which combines the information of the former layers (Zhang , Wang, Zhang , Xu , & Chen, 2019). In the fully connected layer, also known as the dense layer, all possible connections from layer to layer are present, meaning that every input of the input vector affects every output in the output vector. The number of fully connected layers and neurons in these layers varies depending on the architecture (Arora, Garg, & Gupta, 2020).



Picture 12 A visualization of a fully connected layer **(Kost;Altabey;Noori;& Taher, 2019)**

The fully connected layer represents the neural network part of a convolutional neural network where deep learning-related methods such as backpropagation are applied (Ramsundar & Zadeh, 2018). These were discussed in chapter 2.4.

2.5.6 Output layer

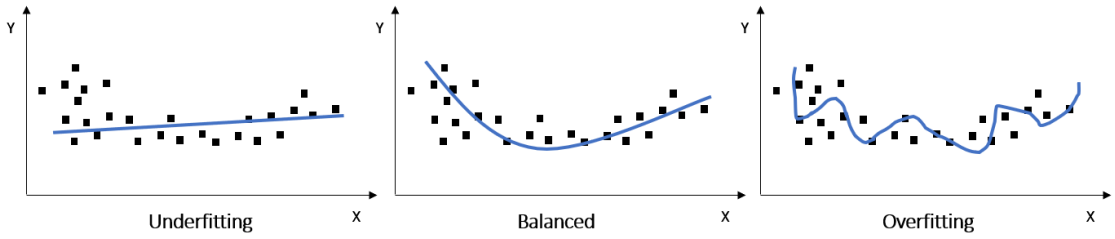
The last layer of a typical convolutional neural network architecture is a fully connected SoftMax layer that computes the network's output. The output is essentially a score probability for each defined class (Zhang ;Wang;Zhang ;Xu ;& Chen, 2019). The SoftMax function used to calculate the probability is a mathematical function that converts the last layer of numbers in the CNN into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The probability is calculated with function 3 where $\vec{z}$ is the input vector, z_i is the input vector's element values, and K is the number of classes (Wood, 2023).

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (3)$$

2.6 Overfitting and underfitting

Two major factors affecting the performance of a machine learning model are overfitting and underfitting. The general purpose of machine learning is to build a model that can fit the training data and make correct predictions for untrained data samples not only the training data. Overfitting and underfitting have both an impact on the model's ability to make correct predictions for samples outside the training data. Minimizing both factors can be quite challenging since they both influence each other (Li;Yan;& Xu, 2021).

Overfitting means that a machine learning model starts to learn or memorize random regularity in the training dataset. The model might perform well on the training data but is not performing well on evaluation data since it is unable to generalize unseen training data. Underfitting is the opposite of overfitting and occurs when the model is incapable of learning features from a training dataset and therefore performs poorly on the training data (Amazon, 2023).



Picture 13 Graphs visualizing under fit, balanced, and overfitting of a machine learning model **(Amazon, 2023)**

Even though, overfitting and underfitting are known challenges and are often warned about in books and research written about machine learning actual theory related to this topic remains relatively underdeveloped. Theory related to overfitting and underfitting relies mainly on general knowledge gained in practice instead of any official set of criteria that would determine whether an algorithm will overfit or underfit a given dataset (Bashir;Mantanez;Sehra ;Segura;& Lauw, 2020).

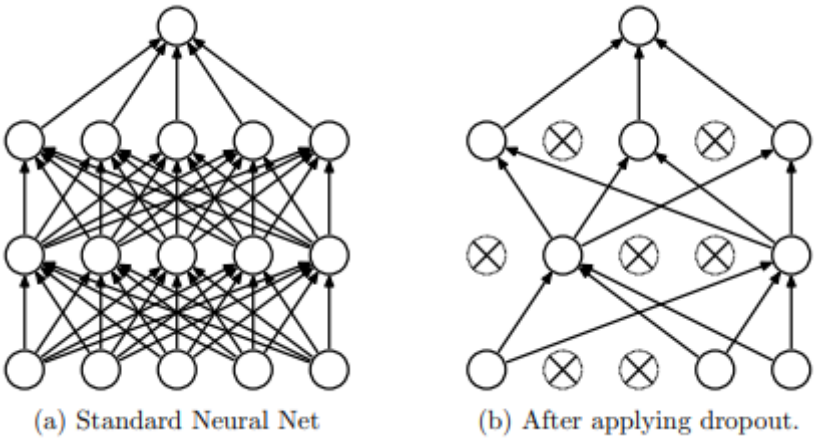
Minimizing both overfitting and under fitting is hard and over time many general methods have been proposed in different research to overcome this challenge without changing the general architecture of the model by increasing or reducing the number of neurons in the network (Li;Yan;& Xu, 2021). To understand how to avoid overfitting and underfitting and how machine learning models are optimized in general this chapter discuss a few parameters that have an effect on the performance of a model and are adjustable: the optimizer, learning rate, dropout, regularization, and data augmentation.

2.6.1 Learning rate and optimizer

Learning rate is one of the most important hyperparameters and has significant effect on the model's ability to converge to an optimal solution. Determining the optimal learning rate happens usually with trial and error during the model's training face. The learning rate is usually related to the optimizer algorithm meaning that different optimizer algorithms have different optimal learning rates (Li;Yan;& Xu, 2021). Optimizers were discussed in chapter 3.4.9.

2.6.2 Dropout

Dropout is a simple method developed to prevent a machine learning model from overfitting. Dropout is at its simplest form a technique where random neurons or units including all connections are dropped from a neural network during the training. According to a journal article "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (Srivastava;Hinton;Krizhevsky;Sutskever;& Salakhutdinov, 2014) this method improves the performance of neural networks on supervised learning explicitly in vision related tasks by reducing the networks sensitivity to certain features and improving the networks generalization ability.



Picture 14 A neural net before and after applying dropout **(Srivastava;Hinton;Krizhevsky;Sutskever;& Salakhutdinov, 2014)**

Dropout is applied by specifying a dropout rate which then determines the number of neurons that are dropped out. The optimal rate is found with a trial-and-error method as are many other hyperparameters in deep learning and machine learning in general (Li;Yan;& Xu, 2021).

2.6.3 Regularization

Another common method for preventing overfitting when training machine learning models is regularization. Regularization is a method where an additional penalty is added to the loss function (chapter 3.4.5) resulting in a sparse parameter matrix that reduces the possibility of overfitting and improving the model's generalization ability (Li;Yan;& Xu, 2021).

There are two common regularization methods L_1 and L_2 regularization where both regularization methods impose a penalty on the magnitude of the model weights in a similar manner (Kamalov & Leung, 2020). The output of the loss function is calculated with function 4 where X is the input value, y the output value, ω is the weight matrix, Ω(ω) is the penalty term, and α is the rate of regularization.

$$\tilde{J}(w, X, y) = J(w, X, y) + \alpha\Omega(w) \quad (4)$$

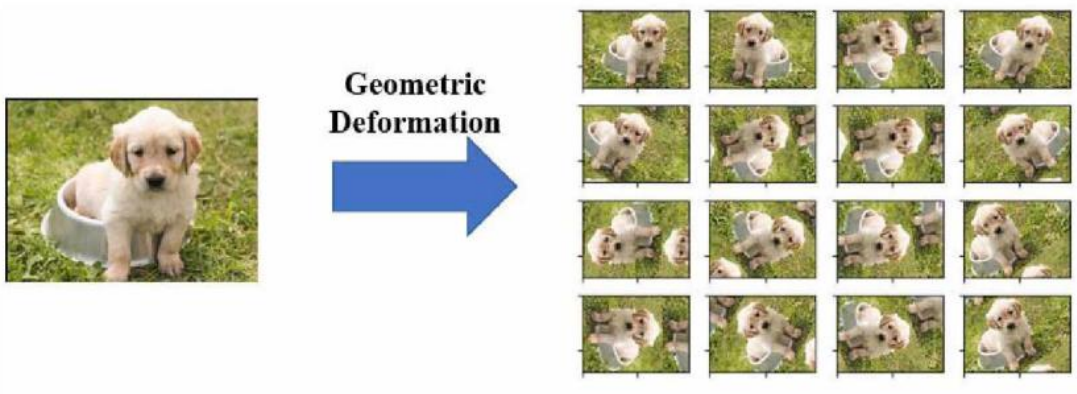
L_1 regularization penalized the weights with function 5 where Ω(ω) is calculated based on the absolute value of ω_i, and L_2 regularization penalized the weights with function 6 where Ω(ω) is calculated based on the squared value of ω_i.

$$\Omega(w) = \|w_1\| = \sum_i |w_i| \quad (5)$$

$$\Omega(w) = \|w_2^2\| = \sum_i w_i^2 \quad (6)$$

2.6.4 Data augmentation

To train a well performing computer vision model with supervised learning requires a sufficient dataset of labelled images. In practice, it is often difficult to obtain enough high-quality images, especially diverse classes of images with labels. Data augmentation is a method that tackles this problem by generating more data from the existing dataset in order to improve the model's performance and avoid overfitting (Yorioka;Kang;& Iwamura, 2020).



Picture 15 Data augmentation achieved with Geometric deformation **(Yorioka;Kang;& Iwamura, 2020)**

There are several different data augmentation methods that can be divided into six different categories image transformation augmentation, image mixed augmentation, feature space augmentation, semi-supervised augmentation, virtual image generation, and intelligent image augmentation. All the methods belonging to these categories have a certain feature or attribute they modify when be applied to a training dataset (Li;Yan;& Xu, 2021). In practice these methods make copies of an image that is part of a training dataset and modify different parameters such as the brightness, contrast, saturation, size, and the angle of an image. A big benefit of data augmentation is that it does not only improve the model's performance and helps avoid overfitting but also saves time and costs on data collection and data labelling (Shah, 2023).

2.6.5 Error analysis

A more methodological approach for detect overfitting and under fitting is to compare and analyse the error generated by the model between a training dataset and a separate test or validation dataset. The difference between these datasets is covered in more detail in the next chapter. In practice the comparison means that the model is trained on a specific data set and the errors it produces on that data is compared to errors on unseen data. When the number of errors in the inferences of a model decreases to a stable level on a training data and simultaneously errors on a separate test data set also behaves in the same manner, this usually implies that the model has generalized the data well without overfitting or underfitting. However, if the errors do not decrease on a training dataset nor on a test data set this is most likely due to underfitting where the model has not been able to generalize all required information from the data. Contrarily, if the number of errors continues to decrease on the training dataset but not on the test dataset, it is a sign of overfitting (Li;Yan;& Xu, 2021).

2.7 Data collection

Data collection is one of the most important and critical parts of a computer vision application development process. According to a survey made on data collection for

machine learning, this has recently become a critical issue since the amount of labelled data is usually not sufficient for the developed machine learning application, especially in deep learning. One of the most difficult processes in classical machine learning is feature engineering, where the user must comprehend the application and offer features for training models. Instead of having to manually create features, which is a crucial aspect of data preparation, deep learning can generate these features automatically. But in exchange, deep learning needs a larger labelled dataset (Roh;Heo;& Whang, 2021).

Data collection can be thought of as a process consisting of the actual data gathering, data labelling, and possibly data acquisition by data augmentation or discovery (Roh;Heo;& Whang, 2021). Gathering the data can be either done by recording with a camera or using existing public datasets. Labelling or annotating data is essentially a process where raw data, i.e., images or video, is annotated by specifying the context with labels that specify which data vectors the model must use for training with software intended for this (IBM, 2023).

2.7.1 Datasets

The collected data is typically divided into training data, validation data, and test data. A training dataset is a dataset consisting of labelled examples of the eventual domain of application used to train the model to fit the training data whereas validation data is data used to validate the performance of the model (Fisher, et al., 2014). According to Brownlee (2023) the terms “validation set” is used interchangeably with the term “test set” both usually referring to the data which is held apart from the training and used to optimize the model’s performance after each epoch. However, validation set is the more common one in this context. Test data can be thought of as an own set of data typically used to perform the final validation of the model’s performance after completing the training by making sure that it is able to generalize well to unseen data (Baheti, 2023).

Split ratio between training, validation, test data really depends on the number of samples present in the dataset and the model. However, there is not really an optimal percentage that would determine how big each data set should be. Common ways to split the data is sixty to eighty percent training data and ten to twenty percent validation and testing data (Baheti, 2023).

3 Project plan

Studies have proved that sufficient project planning is vital to a project's success rate, especially when working on a software development topic where projects tend to fail due to insufficient planning (Serrador, 2013). A good project plan contributes toward a better outcome and further helps conduct this master's thesis. A project plan for the intended computer vision application master's thesis is laid out as a software requirements specification, also known as an SRS document, commonly used in software development projects (IEEE, 1984).

3.1 System features and requirements

When the automated guided vehicles (AGVs) start operations in new areas, new issues may arise that were not previously anticipated. Therefore, the requirements for this master's thesis are defined based on the known issues at the time of conducting the study. The performance of the system is constantly evaluated based on the available data which might bring new insights of the system's performance and potential areas for improvement may emerge. Hence, the focus of this thesis will be to address the challenges and issues currently known and to develop a solution to improve the AGV system's performance, based on the latest available data and best practices for the target company.

3.1.1 Premises and functional requirements

As explained in the introduction chapter, the current safety scanners in the lower part of the AGVs create a two-dimensional plane around the AGV, leaving a blind spot beneath and above the plane. One of the safety scanners is located in the front end on top of the AGV, creating a two-dimensional plane at a 45-degree angle towards the ground. Whenever an object breaks the laser beam within a specific range from a moving AGV, it causes the AGV to stop if the object is not removed. This means that everything that does not break the laser beam goes undetected. E.g., when forks of a forklift left on the

side of the AGVs path break the safety scanner's laser beam, the AGV slows down and stops. However, the breaking does not happen instantly, meaning that the beam has moved forward enough for the forks not to break the beam anymore. Thereby, the AGV does not detect anything in its path and continues driving forward, eventually crashing into the forks. This applies to all objects that break the laser beam only for a second due to their shape or placement.

Another common issue is that the AGVs drive too close to operating manual forklifts before stopping leaving little to no clearance for the manual forklift to turn and move out of the way. This could be solved by having remote controllers in all the manual forklifts operating in target company's facilities that can be used to stop the AGVs. Having a computer vision application that can identify manual forklifts enables the implementation of an automatic system that stops the AGVs earlier when a manual forklift is detected.

Based on the explained premises the minimum requirement for the application is that it would feature an object detection model that detect objects in the AGV's path that it is not able to detect with it current laser scanners. In addition, all manual forklifts should be identified, for the purpose of leaving more clearance between the trucks. For the application to be functional, it must distinguish between built-in structures and movable objects. E.g., when the AGVs navigate between pallet racks in the warehouse, the racks should be identified as built-in structures and therefore ignored. However, since the premise is that an object detection model does only detect objects, it is trained to detect, this is self-explanatory. The objects which the model is trained to detect and classify can be determined with a flowchart presented in figure 1.

If the current system does not recognize an object and there is an imminent risk for the AGV to encounter it and possibly crash into it when operating in the warehouse or production area, the computer vision application should recognize it and signal the AGV to stop. If the current scanners do not recognize an object but there is not a risk that the

AGVs would encounter it, the object should then be ignored as well as if the current system can detect it with its laser scanners.

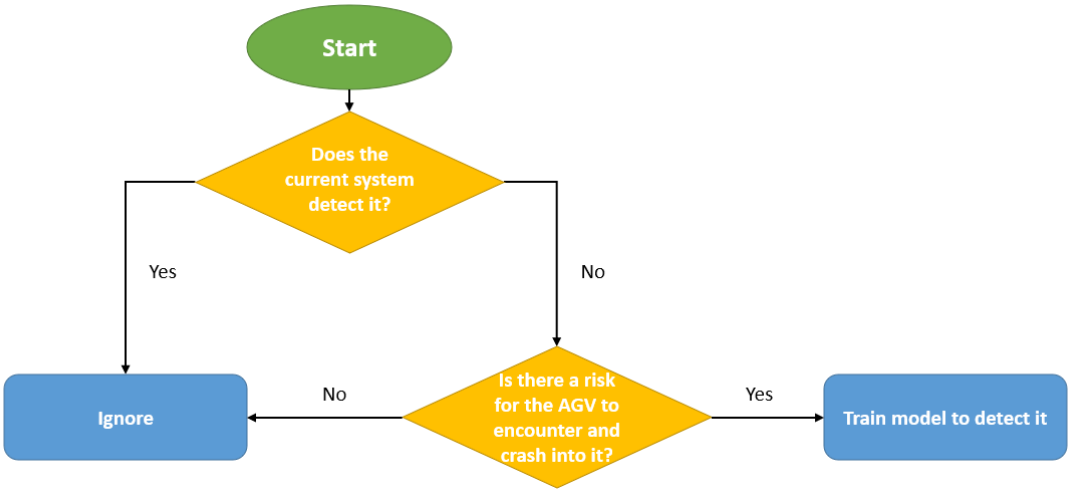


Figure 2 Process flowchart to determine detectable objects

A list of objects that the computer vision application needs to recognize can be seen in table 2. This was acquired by collecting a comprehensive list of objects that the AGV could encounter when operating in target company's facilities and processing each object with the process flowchart presented in figure 1. The complete list of objects is added to appendix 1.

Object	**Imminent risk**	**Trained**
Pallet Jack	Yes	Yes
Rider Truck	Yes	Yes
Reach Truck	Yes	Yes
Counterbalance forklift	Yes	Yes

Table 2 List of objects that the model is trained to detect and classify.

The application should signal the AGV to stop when an object is detected in the AGVs path within a certain distance of the AGV. The transmitted signal would consist of a command for the AGV to stop and possibly information about whether it was a manual forklift. As a real-time application, the delay between detecting an object and sending the signal should be minimal. The top speed of an AGV is approximately 1,2 m/s in the warehouse and factory area and 2,0 m/s in the hub way connecting the warehouse and factory. The camera module's field of vision is about 4,00 meters. In order to determine the distance for when the AGV needs to start braking, the braking distance of the AGVs was measured experimentally for different speeds. The results are visible in figure 2 and 3 where the braking distance is plotted against the speed when using "soft stop" and emergency break and compared to the field of view.

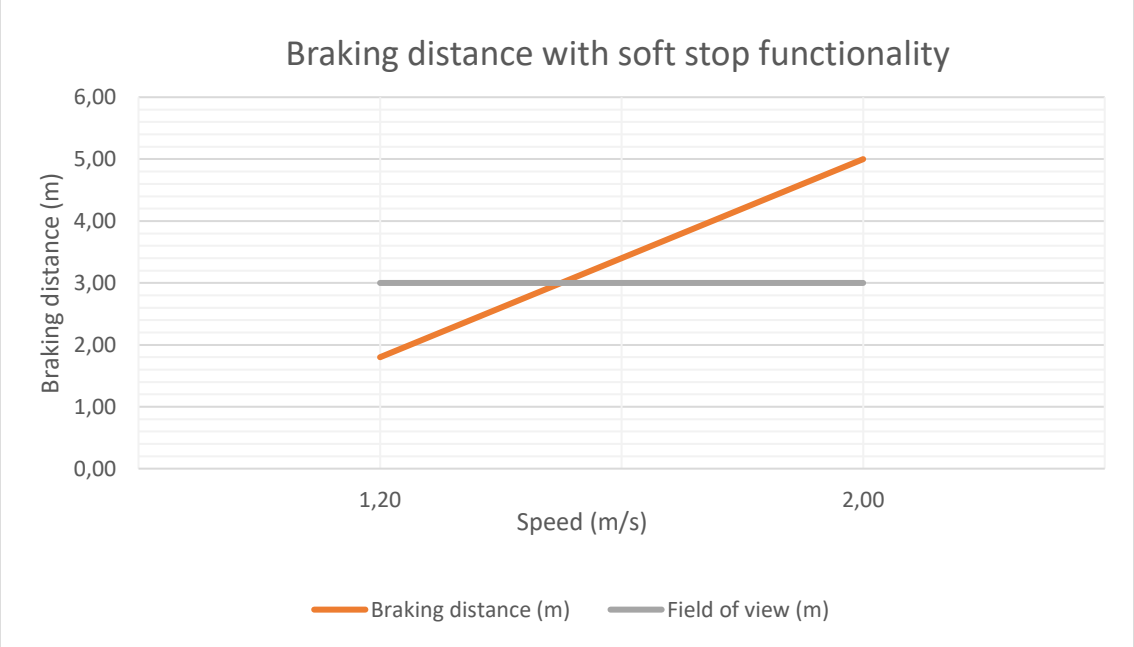


Figure 3 Braking distance-speed graph for soft stop

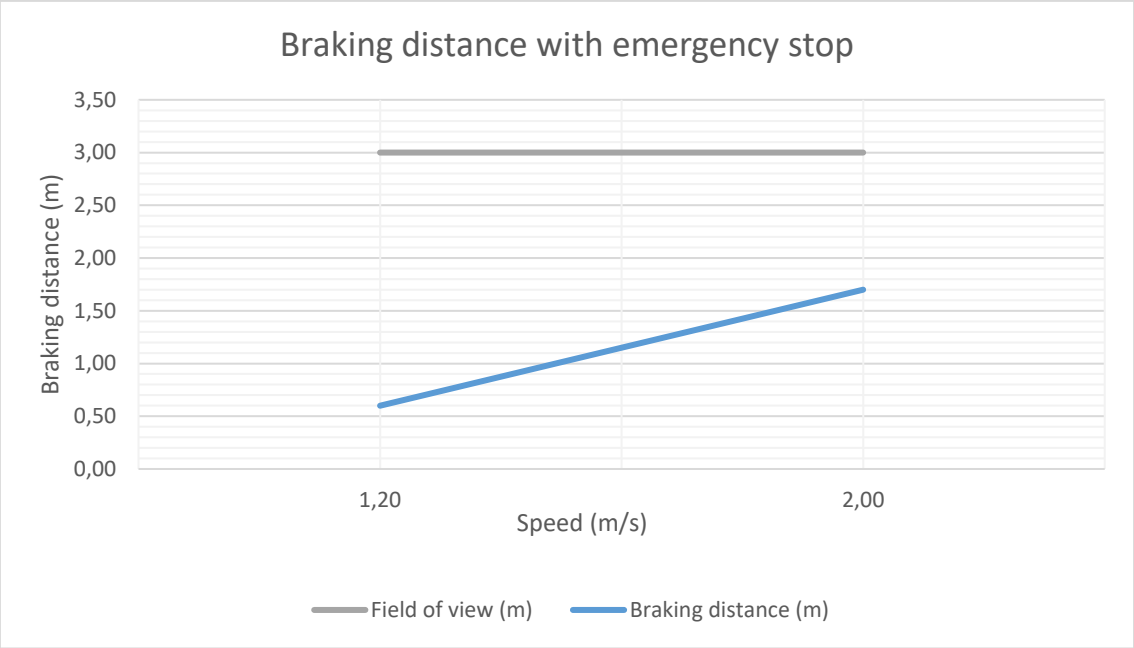


Figure 4 Braking distance-speed graph for emergency stop

Soft stop is a functionality which purpose is to disengage an operating or stationary AGV from the system until it is re-engaged. The AGVs have one soft stop button located on both sides. Emergency stop is essentially as soft stop but stops a moving much faster and requires an additional reset of error messages when pressed. There are altogether four emergency stop buttons on each AGV.

When using the stop functionality, the braking distance is between 1,80 m and 5,00 m, depending on the speed. The AGV's braking distance is 0,6 m to 1,7 m when using the emergency brake functionality. When using the soft stop functionality, the braking distance exceeds the field of view when the speed of the AGVs exceeds approximately 1,5 m/s with no delay in the system. This implies that the AGVs need to brake harder when speeds exceed about 1,5 m/s.

Based on the tests, having a load on does not significantly affect the braking distance when using the soft stop functionality. Therefore, stopping the AGV with the soft stop functionality is adequate when operating in the warehouse and factory area. Based on the tests and calculations, the minimum requirement is that the system can detect an

object and send the signal within the time *t* from when the object appears in the field of view. This is calculated with equation 7 where *F* is the field of view *v* speed and τ delay of the system.

$$t = \frac{F-((4v-3)+v\tau)}{v}, (0 < F \leq 3, 0 \leq v \leq 1{,}5 \; and \; 0 \leq \tau \leq t) \quad (7)$$

Since the braking distance exceeds the field of view when exceeding 1,5 m/s, a functionality for activating emergency braking is required for the AGV to stop before crashing. However, since the AGV exceeds the speed of 1,5 m/s only in the hubway, which is only used for traveling between the warehouse and factory with forklifts, the probability of encountering unexpected obstacles there is relatively low. In addition, determining the AGV's speed accurately for calculating a satisfactory threshold for activating the emergency braking would require additional external equipment. For these reasons having the computer vision system activated in the hubway would not be feasible at this time.

Since the braking distance is 0,6m when using emergency break, all obstacles that appear in the AGV's field of view within 1,8 m from the AGV would be avoided by activating the emergency break in these cases. The activation should happen within the time t_2 from when the object appears in the field of view. This is calculated with equation 8 where *F* is the field of view *v* speed and τ delay of the system where *F* is the field of view *v* speed and τ delay of the system.

$$t_2 = \frac{F-((1{,}375v-1{,}05)+v\tau)}{v} \quad (8)$$

3.1.2 Non-functional requirements

The ability to identify all possible objects that could appear in the AGV's path is not implemented in this master's thesis due to the immersive amount of data required to train a model for that purpose. The requirement is that the object detection model can detect all objects not recognized by the current safety scanners and would cause an eminent safety risk if crashed into.

The current safety scanners are used as the primary system, and the application developed in this master's thesis works as a backup. I.e., if the safety scanners detect something in its path it will stop even though the computer vision application does not detect anything and vice versa. This way the detection of obstacles in the AGV's path is not dependent on only one system.

To meet target company's data security requirements and General Data Protection Regulations set by the EU, none of the video material filmed by the computer vision application is saved on the device itself or anywhere else. The device is not connected to any network and cannot be accessed with a remote connection which ensures the data security of the device. However, the benefits of saving the video footage are discussed but not implemented since developing a prototype of the computer vision application for this master's thesis is still in the face of concept testing.

3.1.3 External interface requirements

The requirement for the object detection model is that it can be implemented on an advanced AI-embedded system to meet the requirement of a real-time application. One possible solution for this could be Nvidia's Jetson embedded platform which is one of the top platforms used for autonomous machines and other embedded applications. The Jetson Nano developer kit is a powerful embedded platform capable of running multiple neural networks in parallel for a computer vision application with relatively low power consumption (Nvidia, 2022).

Sending the signal to stop the AGV could be implemented by replicating a remote controller which is used to soft stop the AGVs. This would essentially be an electronic RF transmitter module connected to the embedded platform. Electricity for powering the system would be provided with a power supply capable connected to the AGV. The requirement for a embedded system of this nature is 5V and 2A or 5V and 4A if performing high-performance computational tasks (Ximea, 2022).

The device should be both compact and easily maintainable, while also enabling development i.e., it should feature components that are easily replaceable. The device comprises multiple components, including a casing, camera module, embedded platform, RF transmitter module, and a fan for cooling. To facilitate efficient production, the physical casing for the camera module and embedded system may be manufactured using the target company's additive manufacturing laboratory.

3.1.4 Quality attributes

Since this application is a real-time safety system, it should be able to detect all objects in its path with close to one hundred percent certainty. Therefore, the current safety scanners work as the primary system, and the computer vision application as the backup. Utilizing computer vision as part of the AGV’s navigation system should improve safety without affecting the system’s overall performance. In practice, the system should not cause unnecessary stops falsely detecting, e.g., traces in the floor as objects, since this would extend the AGV’s pallet delivery time and decrease the AGV system’s overall performance. I.e., the number of false predictions should be as low as possible.

4 Implementation

This chapter presents all the methods and tools used to carry out the practical part of this master's thesis being the actual training of the object detection model required for the computer vision application based on the requirements specification defined in chapter 2. All methods and tools used were mainly chosen based on a prior computer vision application project implemented earlier on target company's premises. Other factors affecting these decisions were prior experience and available documentation. A short overview of YOLO in general and Yolov7 is presented so that the reader is able to understand the kay factors and metrics used to evaluate the results.

The tools utilized in this section was open-source software, including Label Studio and Jupyter Hub. Label Studio is a web-based platform that facilitates the efficient annotation of large datasets for machine learning applications. Jupyter Hub, on the other hand, provides an interactive computing environment that enables the user to create and share code in a web browser. The object detection algorithm employed in this study is YOLOv7. The selection of these tools was based on their proven effectiveness in similar studies, as well as their accessibility and compatibility with the research objectives.

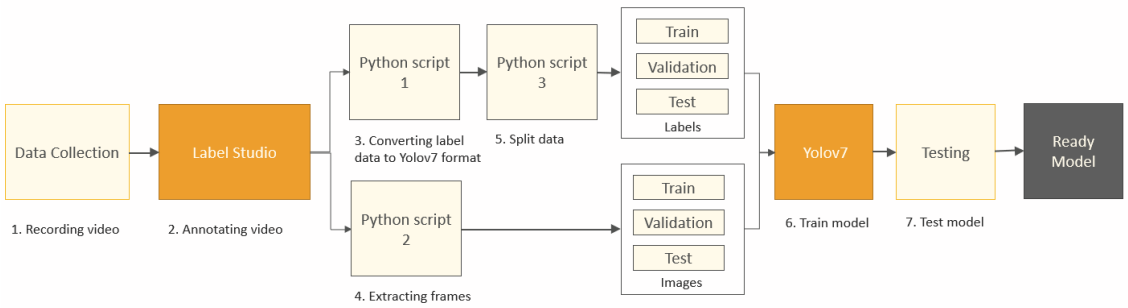


Figure 5 Development process

4.1 Yolov7

YOLO is a convolutional neural network that has evolved from its initial version YOLO to the most recent version of YOLO, version 7 published in 2022. The latest version has several enhancements but shares key concepts the earlier versions. Yolov7 is a real time object detector that according to Wang et al. (2022) surpasses all known object detectors in both speed and accuracy. Yolov7 was selected for this master's thesis since it is a real time object detection algorithm capable of detecting moving objects with high accuracy and is scalable which makes it suitable for e.g., embedded systems. In addition, Yolov7 has delivered promising results in another object detection project implemented on target company's premises where it was used to determine whether a pallet was full or empty (Sormunen, 2023).

4.1.1 Architecture

The general YOLO architecture is composed of a backbone, a neck, and a head. In short terms the backbones main task is to extract essential features from the data set and feed them forward to the head via the neck. The next collects feature maps from the extracted features and creates feature pyramids of them. The head consists of output layers that make the final detections (Chuyi, et al., 2022).

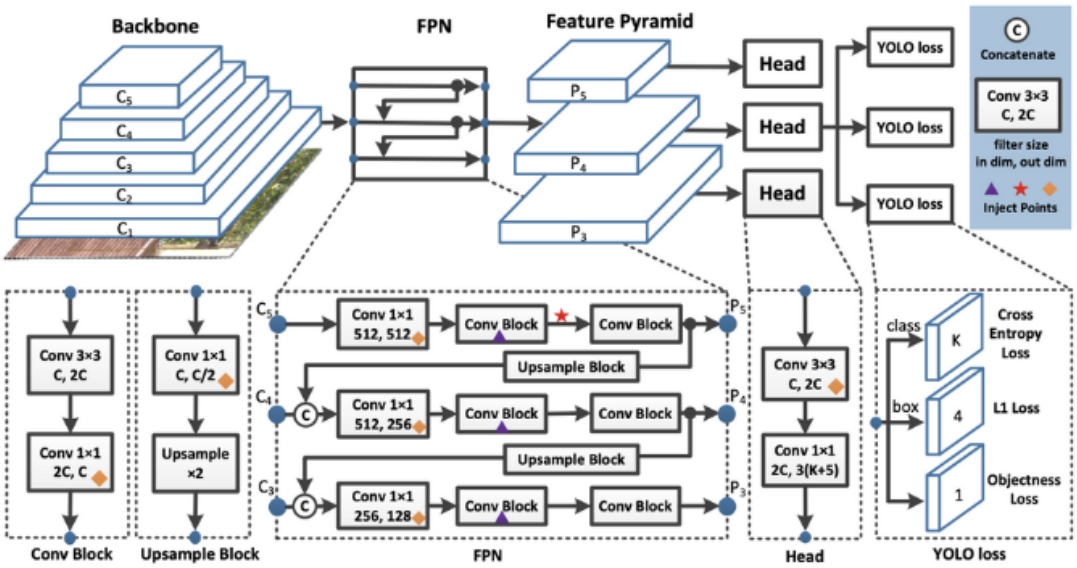


Figure 6 Yolov3 architecture with introduction to PP-Yolo features **(Xiang, ym., 2020)**

YOLOv7 introduces several architectural improvements that increase both efficiency and accuracy. One of the major architectural changes presented in the Yolov7 paper is the introduction of a new computational block E-ELAN (Extended Efficient Layer Aggregation Network) in the YOLOv7 backbone which utilizes expand, shuffle, and merge cardinality in order to improve the learning ability of the network without destroying the original gradient path (Wang;Bochkovskiy;& Mark Liao, 2022).

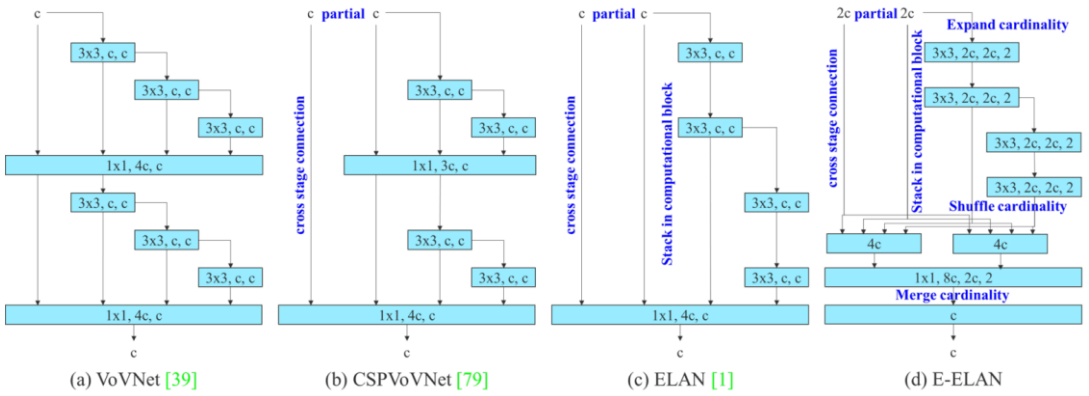


Figure 7 Extended efficient layer aggregation network **(Wang;Bochkovskiy;& Mark Liao, 2022)**

Another major improvement introduced with Yolov7 is model scaling. Model scaling allows to scale an already designed model to fit different type of computing devices by adjusting scaling factors, such as resolution (size of input image), depth (number of layer), width (number of channel), and stage (number of feature pyramid), in order to achieve a reasonable trade-off between the amount of network parameters, computation, inference speed, and accuracy (Wang;Bochkovskiy;& Mark Liao, 2022). This is a noteworthy feature considering the intended end use of the object detection model trained in this master's thesis.

Bag of freebies is a set of techniques or methods that change training strategy or training cost in order to improve accuracy of the model. Bag of freebies was initially introduced

in the Yolov4 paper and can be viewed as a broad framework of training methods for improving an object detection model's overall accuracy. These methods include activations, bounding box regression loss, data augmentation, and regularization method (Chien-Yao, Hong-Yuan, & Bochkovskiy, 2020).

4.1.2 Loss function

The loss function used in Yolov7 is composed of three parts bounding box loss, objectiveness loss, and classification loss (Wang;Bochkovskiy;& Mark Liao, 2022). All these three are modulated by some scalar parameter or IoU score between the model's prediction and a ground truth. Bounding box loss measures the intersection over union between the predicted and target bounding box calculated with equation 9, where Bgt is the ground-truth, and B is the predicted box (Zhaohui, ym., 2020).

$$IoU = \left| \frac{B \cap B^{Bgt}}{B \cup B^{Bgt}} \right| \quad (9)$$

Objectiveness loss measures the objectness which is essentially the probability that an object exists in a proposed region of interest. I.e., when the objectivity is high, the image window contains an object with high probability. Whereas classification loss measures how well the model can predict the correct class of a given object (Kasper-Eulaers, ym., 2021).

On earlier version a loss function called focal loss has been experimented with in order to avoid overfitting. Focal loss is a function that concentrate on the examples where the model fails rather than the ones where it can confidently predict. Focal loss makes sure that predictions on challenging examples get better over time rather than getting too confident with simple ones (Chuyi, ym., 2022).

4.1.3 Metrics used with YOLOv7

Other metrics that are commonly used to evaluate an object detection model's performance is recall, precision, F1 score, and mean average precision. These are all metrics that is given as results by YOLOv7 after a training run.

Recall measures how well the model predicts all existing objects and is calculated with function 10, where TP is true positive, TN true negative, FP false positive, and FN false negative. Whereas precision measures how accurate the predictions are and is calculated with function 11 (Sazanita Isa;Rosli;Yusof;Maruzuki;& Sulaiman, 2022).

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

F1 score is essentially a metric that combines precision and recall measuring the model's accuracy. The F1 score is calculated as the harmonic mean of the precision and recall with function 12 (Lipton;Elkan;& Naryanaswamy, 2014).

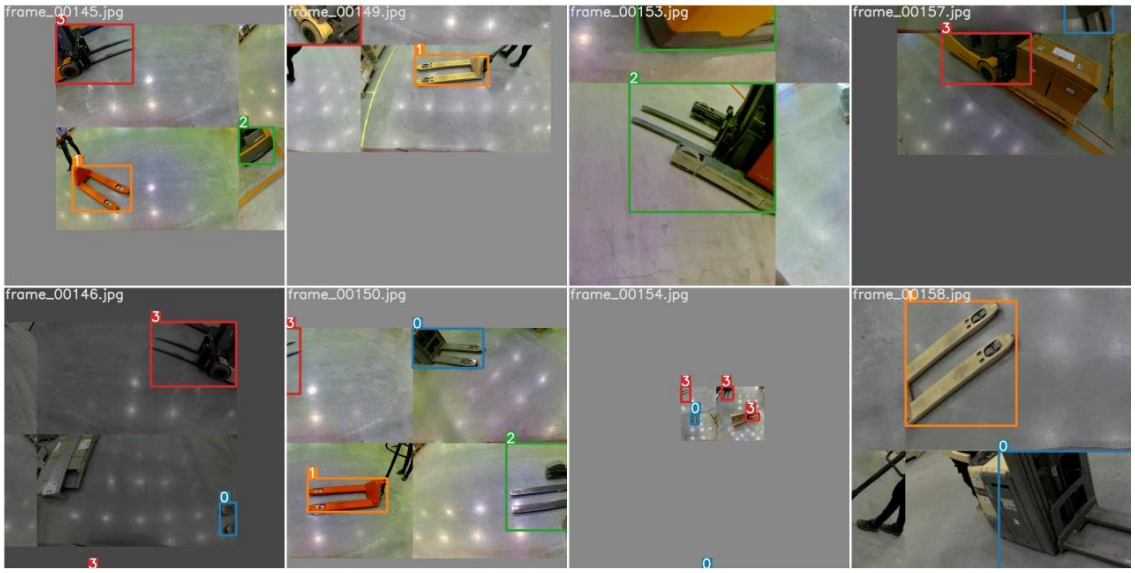
$$F1\ Score = \frac{2TP}{2TP+FP+FN} \tag{12}$$

Mean average precision (mAP) is calculated by finding the mean average precision (AP) over all classes. Mean average precision (mAP) is calculated with function 13, where N is the number of classes and AP the average precision (Sazanita Isa;Rosli;Yusof;Maruzuki;& Sulaiman, 2022)..

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \quad (13)$$

4.1.4 Data augmentation

Mosaic data augmentation was introduced in Yolov4 which is also present in the latest version. Mosaic data augmentation is a method where four training images is mixed which allows the model to detect objects outside their normal context (Chien-Yao;Hong-Yuan;& Bochkovskiy, 2020).



Picture 16 Example of a training batch with mosaic data augmentation used to train the model

Additionally, YOLOv7 has a greater resolution than previous versions by processing images at a resolution of 640 by 640 pixels. This improves the model's capability to detect smaller objects (Wang;Bochkovskiy;& Mark Liao, 2022).

4.2 Data collection

The data used for training the YOLOv7 object detection model was collected by recording the foreground of an operating AGV with a camera attached to the front end of the AGV. The camera was attached approximately to the same spot as the final embedded device would be placed. Majority of the data was collected by filming the AGV's daily operations in target company's facilities and by cutting out clips from the video data with relevant information about the objects defined in chapter 2.3.1. The final scope of objects that were selected to be part of the training set is presented in table 2.

The goal is eventually to gather a comprehensive and expressive enough data set so that the model is able to generalize the different objects in various surroundings. I order to achieve this a set of videos were recorded by intentionally placing objects of each class, on the AGVs path so that they would be visible for the camera. The camera used in the data collection was a Waltter 4k action camera and the videos was taken at 30 FPS with a resolution of 1080x1920 pixels.

A total of 26 separate videos containing 131 minutes of video data was recorded during the data collection process of which 6,7 minutes was used as part of the final training, validation, and testing data sets.

4.3 Data preparation

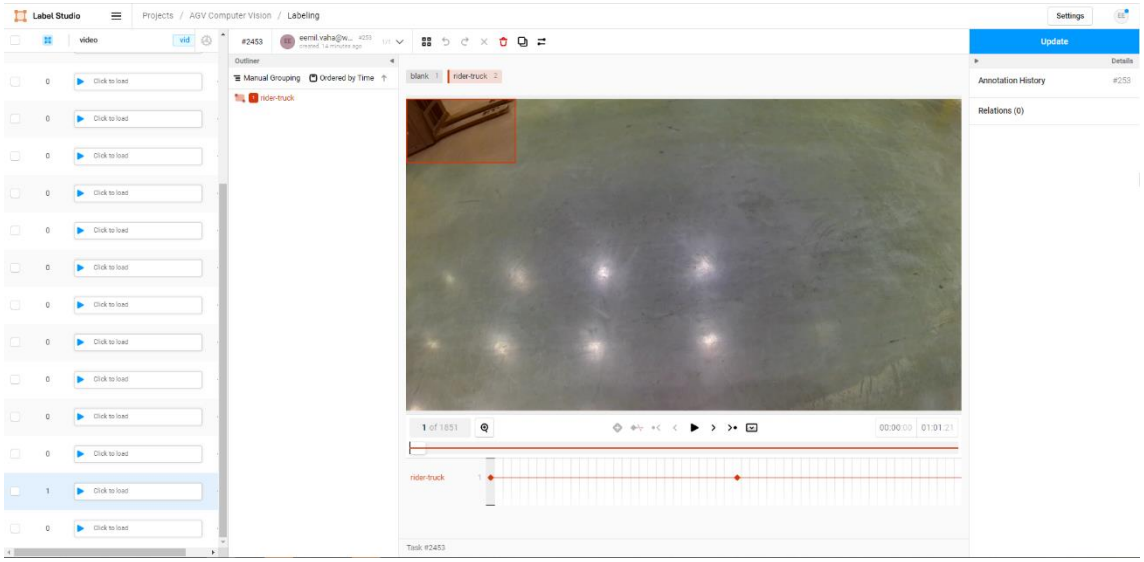
Label Studio which was used to annotate the data only enables to export the annotated video data in JSON format which is not compatible with Yolov7. Therefore, an extra step was required to convert the data to Yolov7 compatible format using a Python script that extracts the information from the JSON file and creates a .txt file containing the data for each annotation. Yolov7 consumes the annotation data from a text file containing information about the class, annotation box place and size in the respective annotated picture. In addition, each frame from the video data had to be extracted and saved as

individual images with the same index as the respective text file containing the extracted frame's annotation data.

The labels and respective images were split into a training, validation, and test set using a python script. The training set contained approximately 70 percent of the data and the validation and test set 15 percent each. To further minimize the potential for overfitting each set contained data from separate videos. The validation data was only used to monitor the model's performance and by having unseen data as validation data gives more reliable results. The total number of annotated images by class in each set is visible in tables 3, 4, and 5.

4.4 Label studio

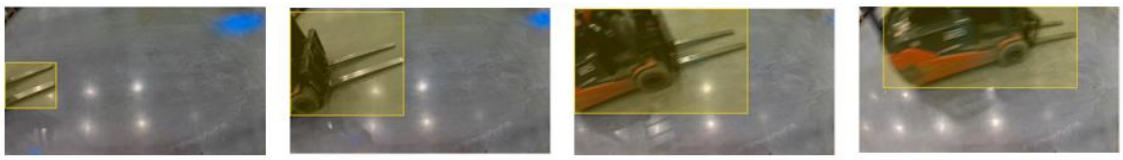
The collected data was annotated using Label Studio, an open-source data labelling platform intended for image classification, object detection, and semantic segmentation. Label studio can be hosted locally but the data was labelled using a target company hosted instance. Label Studio's free version has a simple and easy-to-use user interface with the basic features for image and video annotation which was enough for the purpose of this master's thesis.



Picture 17 Label Studio user interface

Label studio has a feature which allows linear interpolation between frames in a video, i.e., the annotations were done on videos instead of individual images. This sped up the annotation process since a linearly moving object needs to be annotated in only one frame within regular intervals. Each of these annotations works as a keyframe which is used by the Label Studio software to interpolate between the keyframes (Heartex, Inc., 2023). All the data used in this master's thesis was annotated using this feature with an interval of 10 frames.

A general good practice when annotating is to include the target object from corner to corner inside the annotation box. This should be applied consistently to all annotations present in the dataset. In this master's thesis, there was only one person annotating the data which ensured relatively consistent annotation practices which on its own usually results in higher quality labelling data (Liao;Kar;& Fidler, 2021).



Picture 18 Example of a linear motion labelling sequence in Label Studio

4.5 Model training

When the dataset was ready to be used for training the YOLOv7 object detection model the next step was to set up a suitable training environment. To train an object detection model such as YOLOv7 requires an environment with proper libraries installed. The model was trained using a Jupyter notebook an interactive computing platform that was created on a JupyterHub server which is essentially a web-based notebook development

environment that runs in the cloud (Jupyter , 2023). This section covers all the steps required to setup the training environment and the actual training of the model.

4.5.1 Training preferences

The latest version on YOLOv7 was downloaded to the Jupyter notebook from YOLOv7's GitHub repository and all the required libraries was installed using the "requirements.txt" file in the YOLOv7 repository. The notebook was created on a JupyterHub server with a NVIDIA Tesla T4 GPU available for training. All Python commands used to set up the environment are visible in attachment 2.

The goal was to evaluate how well YOLOv7 performs on the given dataset based on a few general evaluation metrics being the classification, objectiveness, and accuracy of the model. The performance was tested by experimenting in a trial-and-error fashion with different parameters for each run with the objective to improve the model's capability to recognizes each class and general accuracy. The parameters that were adjusted between the runs was the batch size, number of epochs, dataset and hyperparameters initial learning rate, final learning rate, and mosaic augmentation.

During this phase the model was trained a total of ten times with three different datasets. The total training time added up to ca. 100 hours. All datasets and training runs with respective parameters are shown in tables 3,4,5, and 6. All other hyperparameters that were not altered during this phase are shown in attachment 3.

Class	**Number of labelled images**		
	Train	**Validation**	**Test**
Rider-truck	3102	1084	602
Total:	3102	1084	602

Table 3 First dataset used for training.

Class	**Number of labelled images**		
	Train	**Validation**	**Test**
Counterbalance-truck	3550	788	1092
Pallet-Jack	2945	856	822
Rider-truck	3102	1084	602
Reach-Truck	3010	1208	645
Total:	15098	3936	3161

Table 4 Second dataset used for training.

Class	**Number of labelled images**		
	Train	**Validation**	**Test**
Counterbalance-truck	3550	788	1092
Pallet-Jack	2945	856	822
Rider-truck	3102	1084	602
Reach-Truck	3500	1208	645
Total:	15608	3936	3161

Table 5 Final dataset used for training.

4.5.2 Model optimization

The initial training run was intended for evaluating how the model behaves on the given dataset and how long the training takes. The first dataset contained only labelled images of one class the rider-truck. After training with the first dataset and default hyperparameters for 10 epochs and a batch size of 9, the results confirmed that the model was behaving as expected and thereby the number of epochs and batch size was increased for the second run with the same dataset.

	Epochs	Batch Size	Initial learning rate	Final learning rate	Mosaic parameter	Mix-up	Scale	Paste in
Run 1	10	9	0.01	0.1	1.0	0.15	0.9	0.15
Run 2	55	16	0.01	0.1	1.0	0.15	0.9	0.15
Run 3	30	16	0.01	0.1	1.0	0.15	0.9	0.15
Run 4	30	16	0.01	0.1	1.0	0.15	0.9	0.15
Run 5	30	9	0.01	0.1	1.0	0.15	0.9	0.15
Run 6	30	16	0.01	0.1	1.0	0.15	0.9	0.15
Run 7	30	16	0.01	0.1	0.5	0.15	0.9	0.15
Run 8	30	16	0.001	0.01	0.5	0.15	0.9	0.15
Run 9	30	16	0.001	0.01	1.0	0.15	0.9	0.15
Run 10	30	16	0.001	0.01	0.5	0.1	0.7	0.05

Table 6 All training runs with respective parameters.

The objectness loss of the model decreased steadily and the precision, recall, mean average precision are increased over time, which indicates that the model can generalize the features of the one class present in the dataset. Based on the these results the dataset was expanded to contain labelled images of 4 different classes. The results of the second training run are visible in figure 8.

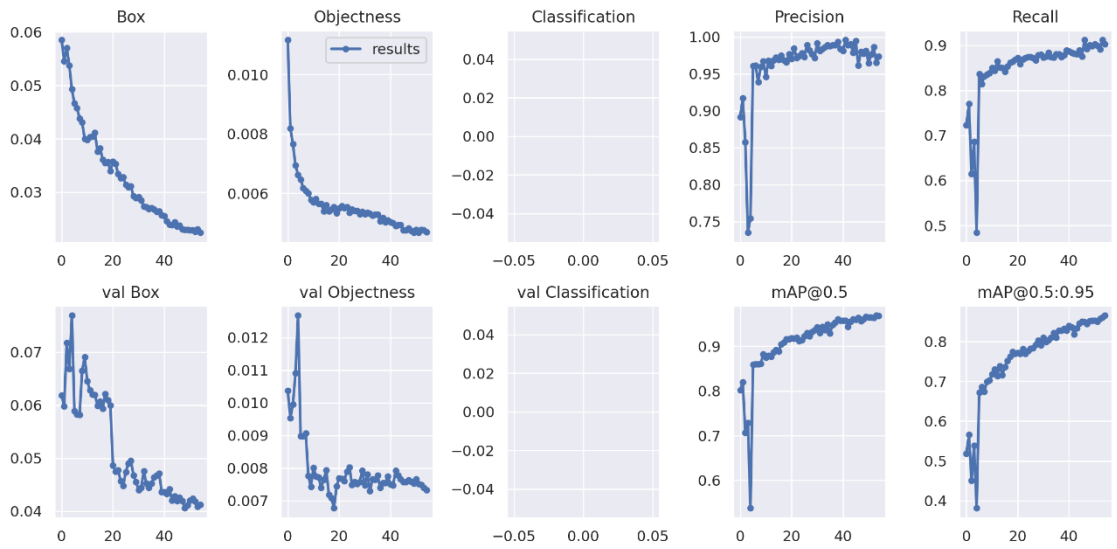


Figure 8 Results of second training run.

In the third run trained with the new dataset containing four classes, the number of epochs was set to 30 and batch size to 16 all other hyperparameters was left as default. The results of the third run on the larger dataset showed some instability based on the precision and recall, however the model seems to converge when looking at the classification loss despite the relatively low number of epochs. The same model was further trained for another 30 epochs with the same parameters which did not have any significant impact on the results. However, the F1 score of the model gives a lower confidence for class “reach-truck”. Based on this information the number of labelled images in the training dataset as well as validation dataset was increased for this class. The fourth training run determined the average training time for one epoch which was approximately 24 minutes. The training was carried out overnight which means that it could not last longer than 12 hours. Thereby, based on this the upper limit for epochs per training run was set to 30 for the rest of the training runs since this is the point where the training time exceeds 12 hours.

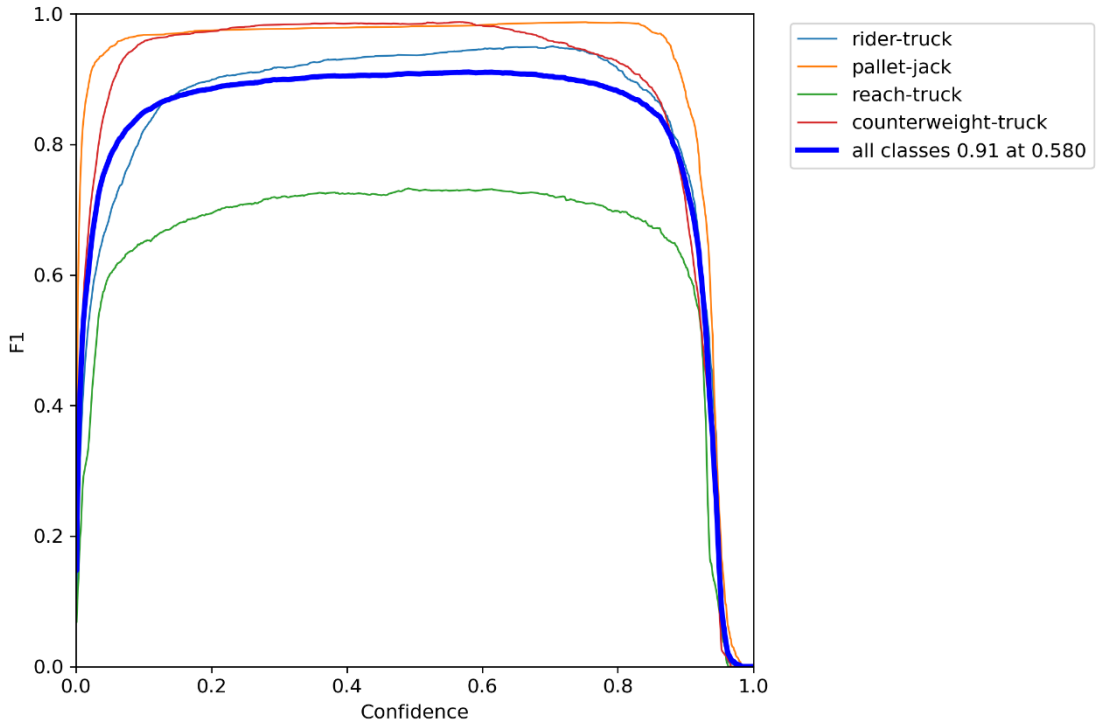


Figure 9 F1 curve after the fourth training run.

The results of training run five and six did show some improvement in confidence for class “reach-truck” based on the F1 score. However, the confusion matrix of the model shows that the model falsely predicts background as reach-truck which might be caused by incorrect labelling. This was checked by manually going through the dataset frame by frame in Label Studio which did not bring out any flaws in the dataset. According to Pham et al. (2022) unrealistic effects which might cause false predictions can be avoided by reducing hyperparameters scale, mosaic, mix-up, and paste in. The effects mosaic augmentation has on the results depends on the complexity of the target objects and background (Wang & Song, 2020). Reach-trucks has more complex features compared to the other classes and would thereby be prone to errors of this nature.

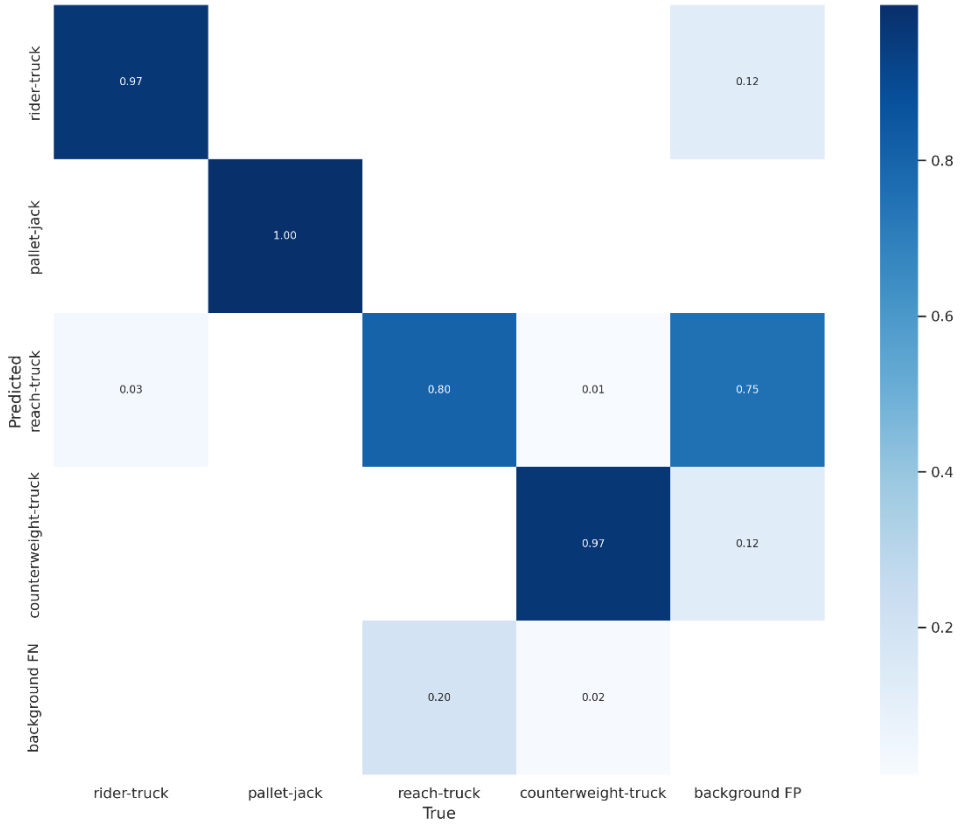


Figure 10 Confusion matrix of training run six.

Lowering the mosaic hyperparameter probability to 0.5 form 1.0 in the seventh training run improved the model's performance on detecting reach-trucks correctly. The model did however show some instability at the start of the training run which isn't optimal when working with a limited number of epochs per training. By lowering the learning rate, it is possible to reduce the oscillation effect (Sazanita Isa;Rosli;Yusof;Maruzuki;& Sulaiman, 2022).

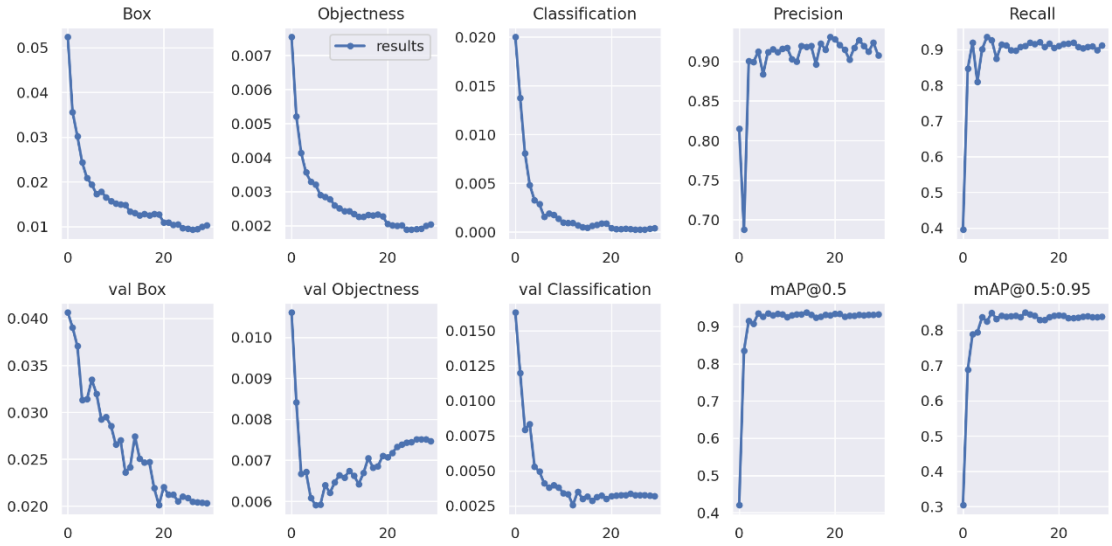


Figure 11 Results of the eighth training run with learning rate and lower mosaic parameter.

Lowering the initial learning rate to 0.001 and final learning rate to 0.01 had a significant effect on the model's behaviour. The model converged a lot faster than in the previous runs when looking at the training loss and mean average precision. The ninth training run was intended to experiment with the initial probability of mosaic augmentation but with a lower learning rate. The initial learning rate and final learning rate was set to the same as in the previous run. The results from this run were in align with the previous results. The model stabilized and converged relatively fast, but some confusion in detecting reach-trucks was apparent in the confusion matrix.

In the last training run the data augmentation parameters mix-up, scale, and paste-in hyperparameters were lowered to see how it affects the model's performance. Lowering the paste-in hyperparameter reduces the training time according to the YOLOv7 documentation (Wang;Bochkovskiy;& Mark Liao, 2022). All hyperparameters of the last training run are visible in attachment 3. Learning rates was the same as in the two previous runs and mosaic parameter was set to 0.5. The results of the last run did not deviate significantly from the previous runs. The model did show some fluctuation in the beginning of the training run for precision and recall as well as for both mean average precision. Overall, the accuracy of the model was at the same level as for the previous runs.

The performance of all models trained in this master's thesis is presented in table 7 where the precision, recall, and mAP@.5 of each model is visible. All other figures related to the results from the training runs mentioned in this chapter are shown in attachment 4.

	Precision	**Recall**	**mAP@.5**
Model 1	0.956	0.826	0.87
Model 2	0.974	0.902	0.968
Model 3	0.9127	0.8899	0.8881
Model 4	0.9316	0.8939	0.9002
Model 5	0.9267	0.9174	0.9197
Model 6	0.9228	0.9099	0.9346
Model 7	0.9353	0.8808	0.9133
Model 8	0.9074	0.9117	0.9335
Model 9	0.9232	0.9192	0.9338
Model 10	0.9193	0.9139	0.9393

Table 7 Overall performance of each model.

4.5.3 Test dataset

The models that have been trained thus far, has only seen data from the training dataset and validation dataset. I order to get more representative results of the models' performance on unseen data the models are tested on the test dataset. Models 1 and 2 are left out since the models was trained only on one class. Models 3 and 4 had a smaller dataset than the rest and are therefore left out. Based on the outcome of this test the best performing model is selected to be further evaluated in a simulated environment with situations that the AGVs most likely encounter when operating in a warehouse environment.

The evaluation of the models' performance is done based on the precision, recall, mAP 0.5, and F1 by class. All models are tested with the same batch size of 16 and the last weights the model had at the end of its training run. The results are shown in table 8 where the highest value in each metric is as dark green, the second highest value light green, third highest as yellow, and lowest value as red.

The best performing model based on the performance on the test dataset is the tenth model with the best overall score compared to the other models. The second-best performing model was the eighth model which did perform much better in recall and precision than the tenth but had lower scores in mean average precision and F1 score than the tenth model. The worst performing model was the ninth model with lowest values in both precision and mean average precision. Having trained the models for only 30 epochs might not show the full potential of all models since some might require longer training time to fully converge due to different learning rates.

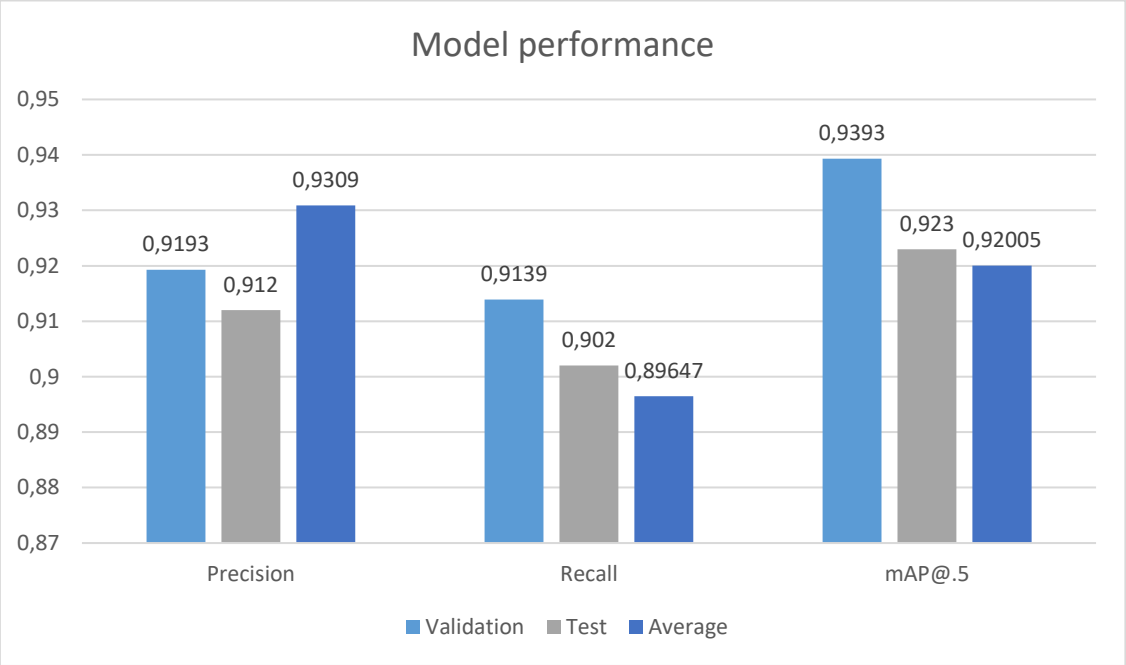


Figure 12 The tenth model's performance on validation and test dataset and average performance of all models.

	Class	Precision	Recall	mAP@.5	F1
Model 5	*All*	0.912	0.871	0.901	0.805
	Rider-truck	0.918	0.987	0.979	0.927
	Pallet-Jack	0.935	0.875	0.875	0.779
	Reach-truck	0.858	0.963	0.947	0.86
	Counterbalance-truck	0.938	0.661	0.804	0.654
Model 6	*All*	0.907	0.901	0.901	0.797
	Rider-truck	0.923	0.971	0.971	0.913
	Pallet-Jack	0.934	0.875	0.873	0.757
	Reach-truck	0.829	0.982	0.936	0.828
	Counterbalance -truck	0.94	0.777	0.826	0.69
Model 7	*All*	0.916	0.876	0.917	0.782
	Rider-truck	0.95	0.947	0.97	0.875
	Pallet-Jack	0.836	0.875	0.857	0.69
	Reach-truck	0.915	0.94	0.949	0.843
	Counterbalance -truck	0.962	0.74	0.893	0.719
Model 8	*All*	0.925	0.903	0.907	0.807
	Rider-truck	0.936	0.989	0.976	0.903
	Pallet-Jack	0.95	0.859	0.874	0.791
	Reach-truck	0.844	0.963	0.936	0.846
	Counterbalance -truck	0.968	0.8	0.844	0.688
Model 9	*All*	0.881	0.898	0.886	0.784
	Rider-truck	0.916	0.977	0.965	0.898
	Pallet-Jack	0.933	0.856	0.84	0.761
	Reach-truck	0.759	0.982	0.924	0.829
	Counterbalance -truck	0.918	0.777	0.816	0.648
Model 10	*All*	0.912	0.902	0.923	0.827
	Rider-truck	0.931	0.974	0.979	0.92
	Pallet-Jack	0.935	0.875	0.868	0.793
	Reach-truck	0.852	0.972	0.973	0.881
	Counterbalance-truck	0.93	0.786	0.874	0.716

Table 8 Test results with the highest scores highlighted with respect to the metric.

Parameter	**Value**	**Information**
Algorithm	YOLOv7	YOLOv7 version from github.com/WongKinYiu/yolov7 on January 2023.
Framework/Library	YOLOR v0.1-121-g2fdc7f1	
Development platform	JupyterHub	All development, training, validation, and testing was carried out on a target company hosted instance of a JupyterHub notebook server
CUDA	Tesla T4, 15109.75MB	GPU resource used to train the model.
Dependencies	See appendix 6	All dependencies are listed in appendix 6.
Hyperparameters	See appendix 5 or chapter 4.5.2	Adjusted experimentally, best performing model's hyperparameters are available in appendix 5.
Optimization	Experimentally	Optimization was carried out experimentally by adjusting hyperparameters based on the model performance (Objectness, classification, precision, and recall).
Validation	Test dataset	The model performance was validated on unseen test dataset. See chapters 4.5.3 and 5.3.

General model architecture	Default	The general architecture of the model such as number of layers, hidden layers, etc. was left as default for the version of YOLOv7 used. See github.com/WongKinYiu/yolov7.
Layers	415	Number of layers in the model used for training.
Gradients	37 212 738	Number of gradients in the model used for training.
Computational complexity	105.5 GFLOPS	Computational complexity of the model used for training.
Optimizer	Stochastic gradient descent (SGD)	Optimizer used for training.
Regularization	Scaled weight decay = 0.0005	Scaled weight decay was not adjusted during training.
Workers	8	Number of workers used during training.
Epochs	See table 6 in chapter 4.5.2	Number of epochs was limited to 30 for all training runs.
Batch Size	See table 6 in chapter 4.5.2	Batch size was limited to 16 due to lack of computational resources.
Classes	4	Number of classes explained in chapter 3.1.1.
Image Size	640x640	Default input image size when using CUDA.
Training data	15608 images (65%)	Consist of images from multiple videos of the objects in the classification scope.

Validation data	4703 images (19,8%)	Consist of images from multiple videos of the objects in the classification scope. Completely different footage than in the training dataset.
Test data	3439 images (0,145%)	Consist of images from multiple videos of the objects in the classification scope. Completely different footage than in the training and validation dataset.
Annotation tool	Label studio v1.7.0	Used to annotate all data used for training, validation, and testing.
Annotation format	PyTorch TXT	Converted from JSON format to PyTorch TXT using a python script and saved in respective files and folder according to YOLOv7 requirements. See chapter 4.3. All scripts available on request.
Image resolution	1080x1920 pixels	All footage used for training was recorded using a Waltter 4k action camera at 30 FPS with a resolution of 1080x1920 pixels.
Model	-	Trained model weights available on request.

Table 9 Summary table of the trained model

5 Testing

To get a better understanding of the trained model's performance proper testing is essential. Object detection model testing differ from traditional software testing and since the concept of object detection is relatively new, principled, and systematic methods for testing object detectors do not yet exist (Wang & Su, 2021). However, testing the object detection model's performance is already performed during the training phase, since the amount of training is based purely on the performance of the model. Therefore, testing in this chapter concentrates on testing feature requirements defined in chapter 3. This chapter covers all steps and results of the testing.

To fully measure how well the selected model can handle situations an operating AGV most likely encounter the model is tested in a simulated environment. Testing is carried out by running the model on a pre-recorded video which is essentially the same as feeding live data to the model. The video contains two different test cases for all four classes. These test cases are incidents that have occurred, close call situations or similar situations that the AGVs most likely will encounter based on the earlier incidents. In addition, the model is tested on how well it can identify a manual truck especially in between pallet racks which was a functional requirement for the application to be able to detect and identify a manual truck to leave more clearance. All test cases are visualized and explained in chapter 5.2.

The model is also tested to evaluate how well it meets the quality attribute that the application should not lower the system's overall performance by causing unnecessary stops with false predictions. I.e., the number of false predictions should be as low as possible. This is carried out by running the model on a video with footage of a regular AGV task with the exception that all instances of objects that the model is able to classify is cut out.

5.1 Test requirements

The model's performance is evaluated based on how well the model meets the requirements defined in chapter 3.3 system features and requirements. The requirements are that the AGV must detect all object specified in the scope of detectable object before the object is under 1,8 meters from the AGV when moving at a speed of 1,2 m/s. Objects that the model must detect and identify are a ride-truck, pallet-jack, reach-truck, and counterbalance-truck. Each scenario or test case has three different evaluation criteria; "Object was detected", "Object was identified", and "Object was identified outside the range of 1,8 m". The result is either "pass" or "fail" for each evaluation criteria and if one of the criteria is a failure the overall test result for that test case is a "failure". To pass a test case all the criteria have to be a "pass".

The line or limit that an object cannot surpass is estimated roughly by placing an object 1,8 m from the AGV and used as reference. Figure 13 visualizes this situation where H_c height of the camera, θ is the pitch angle of the camera, F_w the field of view, and d the distance between an object and the AGV. This method applies only to situations where objects are placed on the ground. However, since the testing is intended to evaluate the overall performance of the model regarding how well it detects object hence the deficiency of not considering objects in the air.

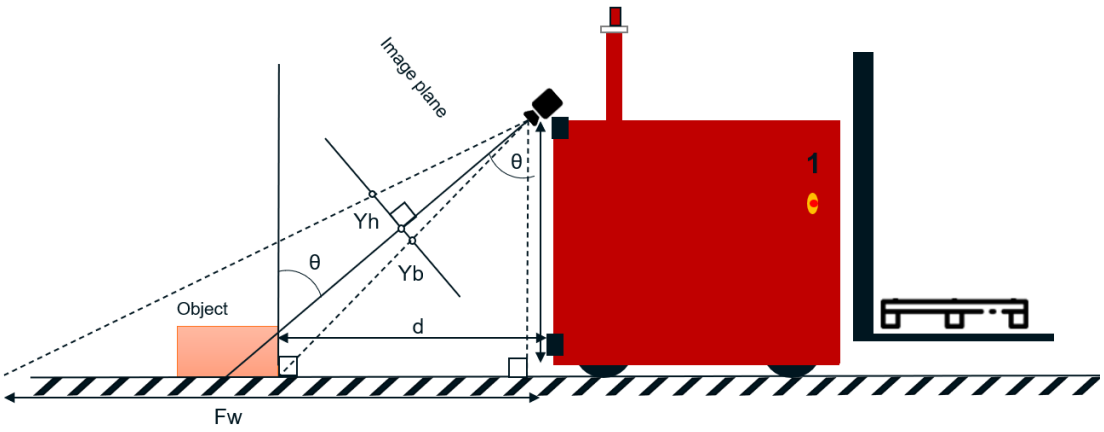


Figure 13 Distance estimation between an object and the AGV.

Picture 19 The horizontal black solid line represents the limit that an object cannot surpass, and the vertical dashed lines illustrates the width of the AGV. A pallet-jack is classified with a confidence of 95%.

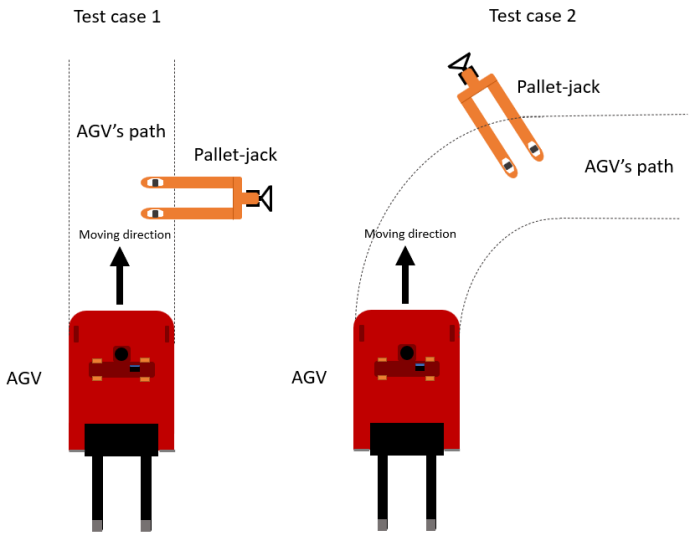
The quality attribute for false predictions is tested by running the model on the video with a confidence threshold of 0.1, 0.3., 0.5, 0.7, and 0.9. The model is tested on 10 000 frames. The results are analysed by counting the number of detections made by the model when the premise is that there should not be any. A detection that lasts for only couple of frames is not worthy of attention since it would not be recognized as an obstacle by the application. Therefore, only detections that lasts for 10 or more frames are considered as remarkable. Additionally, if there is a gap maximum of two frames, it is considered as a continuous detection.

5.2 Test cases

Test cases are selected based on knowledge about what kinds of situations an operating AGV most likely will encounter when operating or incidents where an AGV have crashed into some of the class objects. The situations vary between the different classes since

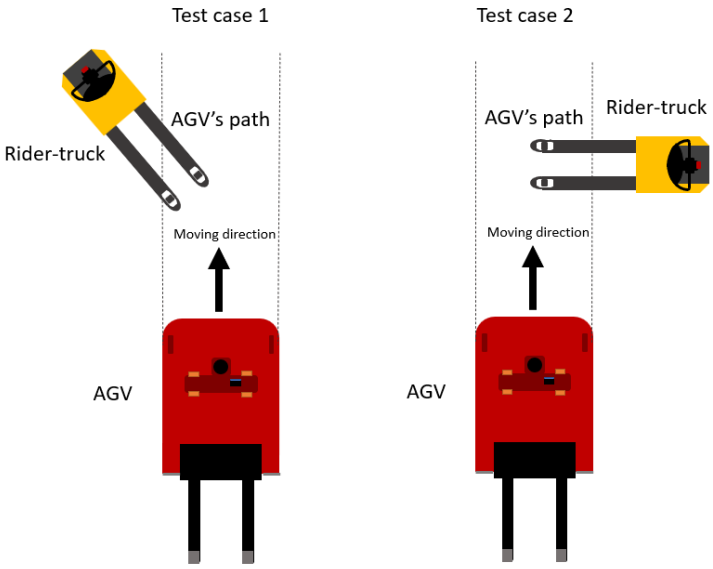
they all are used for different purposes and hence encountered in different places and from different angels. E.g., encountering a stationary counterbalance-truck in the path of an AGV is considerably rarer than pallet-jack or rider-truck.

Pallet-jacks can be encountered basically everywhere from all possible angels since they are mobile equipment and hence tend to be left unsupervised in places they do not belong. AGVs can detect pallet-jacks with their current scanners if the whole pallet-jack is in the path including the shaft. If only the forks of a pallet-jack are in the AGV's path they go undetected. Therefore, the model is tested on scenarios where a pallet-jack is placed so that only the forks are in the AGV's path.



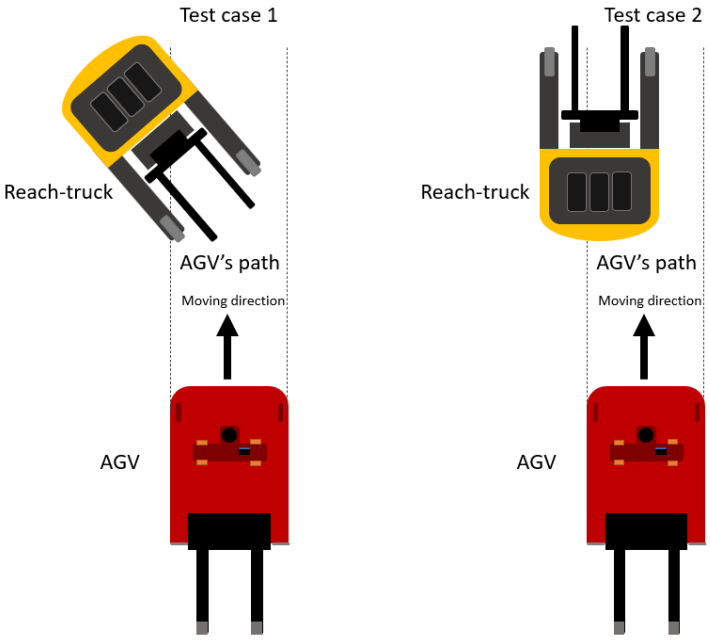
Picture 20 Test cases for pallet-jack.

Rider-trucks are also relatively mobile equipment and often left unsupervised in various places similarly as pallet-jacks. AGVs can detect the body part of a rider-truck with their current scanners but not the forks. Test cases for testing the model on rider-trucks are cases where an AGV encounters a stationary rider-truck with only the forks in the AGV's path.



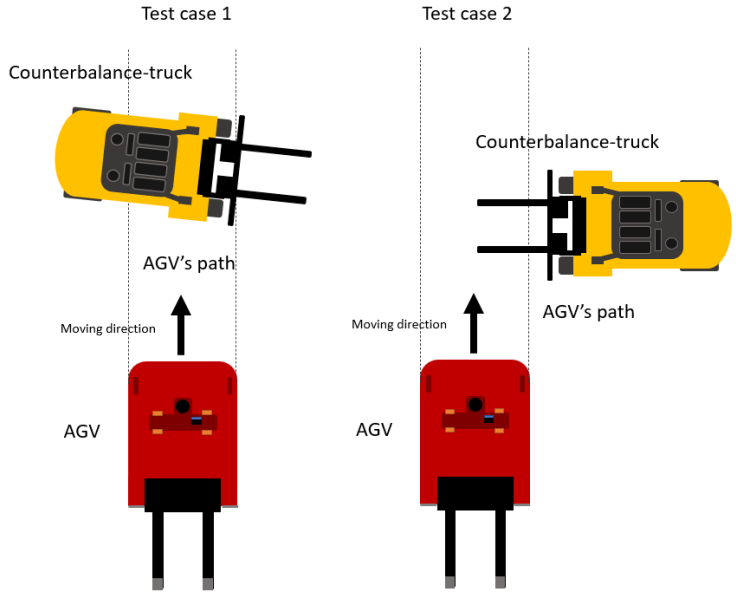
Picture 21 Test cases for rider-truck.

Reach-trucks are most likely encountered between pallet racks when picking or placing pallets. Probability that an AGV would encounter a stationary reach-truck unsupervised in its path is negligible since they are parked in areas where AGVs do not operate. Additionally, a reach-truck operator seldomly must step out of the truck when operating in the areas AGVs operate in. AGVs can detect reach-trucks with their current scanners except if only the forks are in the AGV's path. However, since this scenario is not relevant considering the probability, the model is tested on cases where it should identify a reach-truck from a longer distance to leave more clearance if needed.



Picture 22 Test cases for reach-truck.

Counterbalance-trucks are most likely encountered in the AGV's path when used to move pallets from a place to another or organize pallets on the floor. Counterbalance-trucks are also parked in areas where AGVs do not operate and are therefore rarely encountered stationary and unsupervised. AGVs are also able to detect counterbalance-trucks with their current scanners but not the if only the forks are in its path. Probability that this would happen is higher than with a reach-truck and is therefore tested on the model. The second test case for counterbalance-truck is a scenario where an AGV encounters an operating counterbalance truck from the side which should be identified early to leave more clearance.



Picture 23 Test cases for counterbalance-truck.

5.3 Test results

Results for all test cases defined in chapter 5.2 are presented in table 10. The model passed the tests with a success rate of 100%, which indicates that the model preforms up to expectations. The number of test cases was relatively low but covered all relevant cases considering the requirements. The variation of the background is always very small due to the positioning of the camera, which means that the only notable difference between the different scenarios the positioning of the detectable object relative to the AGV. Having more test cases does not bring any additional value to the testing since it would fall in the category of measuring accuracy of the model, and this was already done when running the model on the test dataset.

The model was able to detect and identify all classes within 19 to 27 frames from when it appeared in the field of view. The video was recorded at a frame rate of 30 fps which means that when the AGV is traveling at a speed of 1,2m/s, there is a 1 second or 30 frame windows for the model to detect and classify the object when the field of view is 3m. However, it is to be noted that the field of view is adjustable by changing the pitch

angle of the camera which gives the application more time to make the detection and sending a signal for the AGV to stop. A screenshot was taken of the moment where the object in question was initially detected and identified and the moment when it crossed the line for activating the brakes for all test cases. All related screenshots are visible in attachment 5.

	Test case 1				**Test case 2**			
	Detected	Identified	>1,8m	OVR	Detected	Identified	>1,8m	OVR
Pallet-jack	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Rider-truck	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Reach-truck	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Counterbalance-truck	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

Table 10 Test results by class and evaluation criteria.

	Test case 1	**Test case 2**
	Time to detect object (frames)	
Pallet-jack	20	24
Rider-truck	27	19
Reach-truck	26	25
Counterbalance-truck	22	24

Table 11 Number of frames it took for the model to detect and classify the objects by test case.

The second requirement for the testing was that the application should not lower the system's overall performance by causing unnecessary stops with false predictions. The results are visible in tables 12 and 13. The model caused a total of 1347 false predictions when running with a confidence threshold of 0.1. Even with a confidence threshold of 0.7 the model detected an object in 191 frames. However, notable is that the number of false detections and duration reduces when the threshold increases indicating that the

predictions have a low confidence. Nevertheless, the model would have caused 8 unnecessary stops when run with a confidence threshold of 0.7 assuming that the detected object is on the AGV's path within 1.8 meters form the AGV.

Threshold	Rider-truck	Reach-truck	Counterbalance-truck	Pallet-jack	Total	Accuracy
Conf. 0.1	899	323	97	28	1347	86,53 %
Conf. 0.3	486	123	38	2	649	93,51 %
Conf. 0.5	190	44	6	0	240	97,60 %
Conf. 0.7	170	19	2	0	191	98,09 %
Conf. 0.9	25	0	0	0	25	99,75 %

Table 12 Results for testing number of false detection in frames.

Threshold	0.3s to 1s	1s to 2s	2s to 3s	3s to 4s	Total
Conf. 0.1	21	5	1	3	30
Conf. 0.3	11	3	2	1	17
Conf. 0.5	5	2	0	0	7
Conf. 0.7	6	2	0	0	8
Conf. 0.9	1	0	0	0	1

Table 13 False detections by duration.

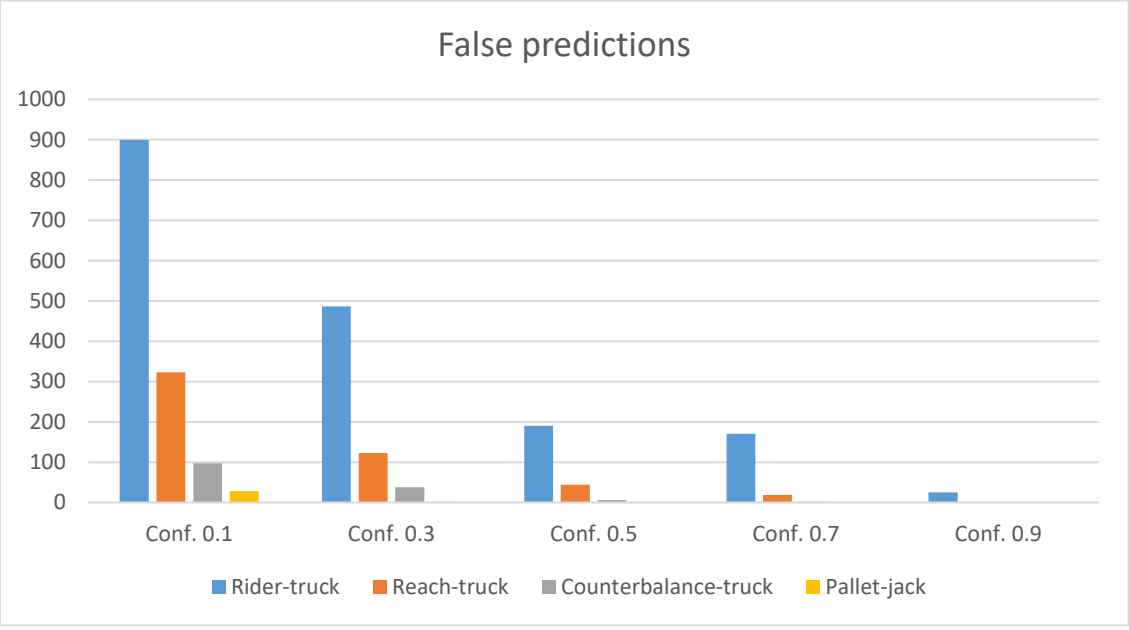


Figure 14 Number of false detections in frames by confidence threshold and object class.

According to the testing results, the model demonstrates its ability to detect and classify all classes in a variety of situations quickly and accurately. It is possible to utilize the model as an obstacle detector for the current classes with a high degree of confidence. Nevertheless, the model's performance is affected by a significant number of unnecessary stops, caused by false positive detections, which decreases its overall usability. Thus, additional training and a more in-depth analysis of the training dataset to address incorrect labelling would be required to enhance the model's performance.

6 Results and observations

This chapter interpret and explain the relevance of the findings in the context of what is previously known about the research problem being studied in this master's thesis, as well as convey all new knowledge or ideas about the issue after taking the findings into account. The objective of this master thesis was to create a proof of concept for a computer vision application that is intended to improve the current AGV system's performance and the research question that this master's thesis attempts to answer is as follows: "Can object detection be utilized to improve the current automated guided vehicle system's performance in terms of avoid crashing into obstacles it cannot detect with its current scanners?".

To answer this question a short literary review of computer vision and object detection was laid out to understand how the technology in question works on theory level. After a general understanding of the concept was acquired, a plan for the computer vision application was laid out in form of a requirement specification in order to properly define requirements the object detection model has to meet. This was then followed by the actual training of an object detection model for the purpose of this master's thesis. Based on the literature it is safe to state that computer vision is a broad concept and has been widely utilized to solve various problems in different business areas and has a lot of potential considering the research problem discussed in this master's thesis. The solution this master's thesis proposes for solving the research problem, focuses merely on object detection in terms of training a model for detecting and identifying a limited number of specified obstacles. The solution is not fully dynamic considering the variety of obstacles an AGV could possibly encounter and not detect, but it supplements the limitations of the current system in a critical area and is easily scalable.

In this master's thesis a YOLOv7 object detection model has been experimentally validated for accuracy and examined as a potential solution to the current incapabilities of the AGV system. Based on the results of this master's thesis, object detection could be applied as a potential solution to improve the current automated guided vehicle

system's performance in terms of avoid crashing into obstacles it cannot detect with its current scanners, when considering the performance of the required object detection model. The best performing model trained in this master's thesis measured a mean average precision of 0.9393 mAP, recall of 0.9139, and precision of 0.9193. The model passed all test cases with a success rate of 100%, which means that it fulfils all requirements set for the application on a theoretical level. The number of classes the model was trained to detect in this master's thesis was only four different classes and having more classes would essentially require more data which would result in longer training times. A limiting factor during the conduction of the master's thesis was the available resources for training the model in terms of computational power. The training was carried out outside regular office hours which limited the training time to 30 epochs for each model. However, scaling the model to detect new object require additional data and it is easily acquired by consistently recording daily operations of the AGVs in the same fashion as done during the data collection phase in chapter 4.2. By doing these new problematic obstacles could be identified from the data and thereby included in the model's training dataset.

As explained the solution this master's thesis proposes is not dynamic in the sense of it being only able to detect obstacles the model is trained to detect. However, a classifying object detector enables profound actions based on the classified obstacles. I.e., when a specific obstacle or object is detected, the application can act accordingly e.g., leave more clearance when a manual forklift is in the AGVs path. As the results of the testing showed, a YOLOv7 object detector can detect and classifying objects fast and precisely by only seeing small portion of the object. In the testing the YOLOv7 object detector measured an average speed of 0.3ms per inference. For the application to be able to make decision based on the classification the model must perform with a high accuracy and should not confuse a specific object for another. The tenth model measured the highest overall F1 score of 0.827 for all classes. Surprising was that the model was measured a F1 score of 0.793 for pallet-jack even though it is a relatively simple object.

Although, the model showed a low F1 score for some of the classes it predicted objects correctly in the test video with a high confidence of 0.9 and higher.

As this this master's thesis has pointed out, data collection and annotation processes are both crucial considering the whole development process. The more consistent high-quality data available for training, the better the outcome is, as can be seen when comparing the results between training runs three and five in figures 14 and 15, where the dataset was increased. This master's thesis did not focus on fixing incorrect annotations or false predictions other than manually checking the correctness of the datasets used to train the model and running the model on a test video to analyse the number of false predictions. The results showed that the model would cause unnecessary stops in its current form due to it falsely detecting objects with a rate of 2.4% when the confidence threshold is 0.5 if the detection would happen on the AGVs path within 1.8m from the AGV. The same rate was 13.47% when running on a confidence threshold of 0.1. The model falsely classified unseen objects, such as a cleaning cart and a door with relatively high confidence of 0.9 and higher. This implies that the model would need to be further trained to increase the accuracy. The lower confidence false predictions were mostly marks in the floor and random pallets in the outer periphery. This could be explained with incorrect annotations or overfitting due to the data being rather homogeneous since data was collected in form of a video where the annotated frames do not deviate a lot from each other. This is also visible in picture 24 where an example of a training batch containing 16 frames that were used to train model number ten. To further improve the model's performance in the future a proper tool intended for analysing the data to eliminate possible incorrect annotations would be advantageous.

This master's thesis did not experiment with the other lighter available versions of YOLOv7 such as YOLOv7-tiny running an edge GPU-oriented architecture optimized for inference on edge devices. The lighter version is expected to measure much faster inference speeds but slightly decreased performance in terms of accuracy

(Wang;Bochkovskiy;& Mark Liao, 2022). However, the proposed Nvidia hardware family can run a YOLOv7 object detector with the higher-end hardware but not suitable for not suitable for mobile device deployments due to the high computational requirements of YOLOv7. Nvidia's Xavier AGX can deliver rates of 17 fps with YOLOv7, which is suitable for a real time industrial object detector for the purpose of this master's thesis. Generally, the inference speed of the device should be 30 to 60 fps for an industrial real time object detector (Nguyen;Bae;Lee;Lee;& Kwon, 2022). Thus, depending on the speed an AGV is traveling and the pitch angle of the camera, it is enough that the device can deliver inference speeds of 12 fps and higher which translates to a travel distance of 10cm and less per frame. Being able to detect obstacles every 10cm is enough for the application to be functional.

Based on the optimization process carried out during the training and the testing performed afterwards, it turned out that decreasing hyperparameters related to data augmentation improves the model's performance when using a dataset like the one used in this master's thesis. In the tenth training run the data augmentation parameters mix-up, scale, and paste-in hyperparameters were lowered slightly. The effects of lowering these parameters where not clearly visible during the training process, but the based on the model's performance on the test dataset it had a positive effect. A speculative conclusion can be made that the nature of the dataset used in this master's thesis deliver better model performance with lower data augmentation effects since the background of the objects is always the same both in the training dataset and end use case. To confirm the findings more comprehensive testing would be required.

The next step in order to implement the actual application would be to investigate the hardware requirements for implementing the communication between the application and the AGVs and piloting the trained YOLOv7 object detection model on embedded hardware. This master's thesis could be used as a reference when defining how the computer vision application should interpret the detections made by the object detector in terms of functionality. The overall idea is clear but for the application to be fully

functional details like handling turns where objects are closer to the AGV would need be managed without affecting the system's ability to detect obstacles. I.e., the application would need to identify that the AGV is turning and thereby narrow down the limit for activating the brakes if an obstacle is detected within this limit.

7 Conclusions

The complex nature of the production and warehouse environment poses a significant challenge to the current automated guided vehicle system. The system's sensitivity to unexpected changes and challenges in the environment results in a slowdown of operations, which cannot be efficiently resolved with static functions. Given the dynamic and constantly evolving nature of the environment, an adaptive solution is necessary to address the multi-dimensional problem at hand. The use of an object detection application with high speed and accuracy has the potential to provide the necessary adaptability to overcome the challenges faced by the current system.

As demonstrated in this master's thesis, preliminary work is crucial for the success of any project. Adequate practical knowledge of the target process, coupled with a comprehensive understanding of the underlying technology, enables the development of a well-performing object detector capable of addressing complex problems such as the one presented in this thesis. Computer vision enable development of more complicated autonomous systems but to fully leverage this technology it is essential for the developer to have a comprehensive understanding of the underlying technology. Ideally, in machine learning, the idea is to select a model at the sweet spot between underfitting and overfitting, or at least this is the goal but is very difficult to achieve in practice. Determining the problem and defining the requirements of the applications are essential but are only a part of the whole process. Collecting sufficient high-quality data for training the model is critical for achieving good model performance, assuming that annotations are performed through a well-defined process to eliminate inconsistency and incorrect labelling.

As a result of the proposed solution for the research problem in this master’s thesis, the AGVs most likely stop more often, but by doing that they avoid crashing into obscure obstacles. With the help of a classifying object detector running on an embedded system it is possible to control the AGVs in a way that would streamline forklift traffic between narrow racking ails and elsewhere in the warehouse and factory areas. The outcome of

this master’s thesis did not only prove the potential and capability of a YOLOv7 object detector but had a positive influence on target company’s employees’ prejudice toward new technology and the AGVs themselves. Since the project was carried out on-site the employees working in the warehouse got hands-on experience on how a system like the AGV system could be enhanced. Most often, employees working on the shop floor have the best view of what the issues are, e.g., considering a manufacturing process or supply chain. As studies on resource-based view of strategy have pointed out, frontline employees’ operational improvement competence and creativity directly correlate with a company’s ability to achieve competitive advantages (Yang;K.C. Lee;& Cheng, 2016). This master’s thesis contributes directly to target company’s employees’ OIC by giving practical knowledge about applied machine learning and computer vision. Meaning that frontline employees of target company have improved premises to recognize potential use cases for computer vision in their line of work.

Although the YOLOv7 object detector showed promising results as a potential solution for addressing the limitations of the current AGV system, it may be worth exploring a more dynamic approach, such as anomaly detection with computer vision, to compensate for the object detector's lack of adaptiveness. Anomaly detection involves identifying events or items that deviate from what is expected, in this case, an obstacle in the AGV's path (Cambridge Dictionary, 2023). In this approach, the AGV's movements are recorded and classified as a clear path. The generalized data would then be compared to a live video feed of the AGV's foreground, and if an obstacle appeared, it would deviate from the generalized data, resulting in detection. This method could complement the proposed classifying object detector and serve as an additional system to improve the AGV's navigation capabilities.

The findings of this master's thesis suggest that developing and training an object detection model can be accomplished with moderate effort, even without extensive expertise in applied computer vision. This observation could have important implications for the target company, as it may encourage them to invest in similar projects that

leverage cutting-edge technologies, such as computer vision and machine learning, in business areas where these technologies have not been applied before. By doing so, the company could gain a competitive advantage in their industry, increase operational efficiency, and improve overall performance.

The object detection model developed in this master's thesis has the potential to be utilized in other perceptual object detection applications requiring accurate classification of model inferences. By following the steps presented in the implementation phase it is possible to train a model that is able to classify objects with high accuracy and possibly even better if further trained with data from a similar environment. The trained model could serve as a foundation for comparable projects, thereby reducing the time and costs involved in preliminary research efforts. When training a similar object detection model, it is crucial to invest on the quality of the training data by applying tools that helps both preparing and validating the correctness of the annotated data. Having tools for this purpose streamlines the data preparation process and saves time from analysing primary causes of unwanted model behaviour. Based on the amount of training time used in this master’s thesis to train a model the configurations of the model should preferably be defined based on other related work rather than relying on experimental optimization from scratch.

The results of the training and testing conducted in this master’s thesis implied that reducing data augmentation parameters on a YOLOv7 model improved the model’s performance when operating in an environment where the background does not vary a lot. On this basis, future research related to object detection should further examine whether less data augmentation improves the performance of an object detection model in similar environments. This could possibly facilitate the training of better performing object detection models with shorter training times using similar data in terms of background features.

References

Amazon. (2023). *Amazon Machine Learning Developer Guide.* Amazon Web Services.

Amitha, M.;Amudha, P.;& Sivakumari, S. (2020). Deep Learning Techniques: An Overview. *Advanced Machine Learning Technologies and Applications, 2021, Volume 1141*, (ss. 599-608).

Amos, B.;Xu, L.;& Kolter, J. Z. (2017). *Input Convex Neural Networks.*

Arora, D.;Garg, M.;& Gupta, M. (2020). Diving deep in Deep Convolutional Neural Network. *2020 2nd International Conference on Advances in Computing, Communication Control and Networking.* IEEE.

Baheti, P. (17. 01 2023). *Train Test Validation Split: How To & Best Practices [2023]*. Noudettu osoitteesta v7labs: https://www.v7labs.com/blog/train-validation-test-set

Bashir, D.;Mantanez, G.;Sehra , S.;Segura, P. S.;& Lauw, J. (2020). An Information-Theoretic Perspective on Overfitting and Underfitting. *AI 2020: AI 2020: Advances in Artificial Intelligence* (ss. 347-358). Springer, Cham.

Bernico, M. (2018). *Deep Learning Quick Reference : Useful Hacks for Training and Optimizing Deep Neural Networks with TensorFlow and Keras.* Packt Publishing, Limited.

Bottou, L. (2018). *Online Learning and Stochastic Approximations.* AT&T Labs–Research.

Brownlee, J. (17. 01 2023). *What is the Difference Between Test and Validation Datasets?* Noudettu osoitteesta machinelearningmastery: https://machinelearningmastery.com/difference-test-validation-datasets/

Cambridge Dictionary. (13. March 2023). *Meaning of anomaly in English*. Noudettu osoitteesta Cambridge Dictionary: https://dictionary.cambridge.org/dictionary/english/anomaly

Campesato, O. (2020). *Artificial Intelligence, Machine Learning, and Deep Learning.* Mercury Learning & Information.

Chien-Yao, W.;Hong-Yuan, M. L.;& Bochkovskiy, A. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection.*

Chokmani, K.;Khalil, B. M.;Ouarda, T. B.;& Bourdages, R. (2007). *Estimation of River Ice Thickness Using Artificial Neural Networks.* Quebec: CGU HS Committee on River Ice Processes and the Environment.

Chuyi, L.;Lulu, L.;Hongliang, J.;Kaiheng, W.;Yifei, G.;Liang, L.;. . . Xiaolin, W. (2022). *YOLOv6: A Single-Stage Object Detection Framework for Industrial.*

Dadhich, A. (2018). *Practical Computer Vision : Extract Insightful Information from Images Using TensorFlow, Keras, and OpenCV.* Mumbai: Packt Publishing.

Davies, E. (2017). *Computer Vision : Principles, Algorithms, Applications, Learning .* London: Elsevier Science & Technology.

Fisher, R. B.;Breckon, T. P.;Breckon, T. P.;Dawson-Howe, K.;Fitzgibbon, A.;Robertson, C.;. . . Williams, C. K. (2014). *Dictionary of Computer Vision and Image Processing.* Hoboken John Wiley & Sons, Incorporated 2014.

Goodfellow, I.;Bengio, Y.;& Courville, A. (2018). *Deep learning.* The MIT Press 2018.

Haji, S. H.;& Abdulazeez, A. M. (2021). *COMPARISON OF OPTIMIZATION TECHNIQUES BASED ON GRADIENT DESCENT ALGORITHM: A REVIEW.* Duhok: Duhok Polytechnic University.

Heartex, Inc. (28. 01 2023). Noudettu osoitteesta Label Studio: https://labelstud.io/

IBM. (17. 12 2022). *What is Artificial Intelligence (AI)?* Noudettu osoitteesta IBM CLoud Learn Hub: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

IBM. (16. 01 2023). *data-labeling*. Noudettu osoitteesta IBM: https://www.ibm.com/topics/data-labeling

IEEE. (1984). *IEEE Guide to Software Requirements Specifications.* New York: The Institute of Electrical and Electronics Engineers.

Jupyter . (19. 01 2023). *JupyterHub*. Noudettu osoitteesta Jupyter: https://jupyter.org/

Kamalov, F.;& Leung, H. (2020). Deep learning regularization in imbalanced data. *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)* (ss. 1 - 5). Sharjah: IEEE.

Kasper-Eulaers, M.;Hahn, N.;Berger, S.;Sebulonsen, T.;Myrland, Ø.;& Kummervold. (2021). Short Communication: Detecting Heavy Goods Vehicles in RestAreas in Winter Conditions Using YOLOv5. *Algorithms 2021, 14*, 114.

Kost, A.;Altabey, W. A.;Noori, M.;& Taher, A. (2019). *Applying Neural Networks for Tire Pressure Monitoring Systems.* ResearchGate.

Kuleshov, V.;& Ermon, S. (2017). *Deep Hybrid Models: Bridging Discriminative and Generative Approaches.*

LeCun, Y.;Bengio, Y.;& Geoffrey, H. (May 2015). Deep Learning. *Nature 521*, 436-44.

LeCun, Y.;Bottou, L.;Bengio, Y.;& Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 2278 - 2324.

LeCun, Y.;Kavukcuoglu, K.;& Farabet, C. (2010). Convolutional Networks and Applications in Vision. *International Symposium on Circuits and Systems (ISCAS 2010).* Paris.

Li, Q.;Yan, M.;& Xu, J. (2021). Optimizing Convolutional Neural Network Performance by Mitigating Underfitting and Overfitting. *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)* (ss. 126 - 131). Shanghai: IEEE.

Liao, Y.-H.;Kar, A.;& Fidler, S. (2021). Towards Good Practices for Efficiently Annotating Large-Scale Image. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (ss. 4348-4357). Nashville: IEEE.

Lipton, Z. C.;Elkan, C.;& Naryanaswamy, B. (2014). *Thresholding Classifiers to Maximize F1 Score.* San Diego: University of California.

Mayershofer, C.;Holm, D.-M.;Molter, B.;& Fottner, J. (2022). LOCO: Logistics Objects in Context. *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA).* Miami: IEEE.

McCarthy, J. (2007). *WHAT IS ARTIFICIAL INTELLIGENCE?* Stanford: Computer Science Department Stanford University.

Nguyen, H.-V.;Bae, J.-H.;Lee, Y.-E.;Lee, H.-S.;& Kwon, K.-R. (2022). Comparison of Pre-Trained YOLO Models on Steel Surface Defects Detector Based on Transfer Learning with GPU-Based Embedded Devices. *Sensors 2022*.

Nixon, M.;& Aguado, A. (2020). *Feature Extraction and Image Processing for Computer Vision.* London: Elsevier.

Nvidia. (15. 11 2022). *Edge Computing*. Noudettu osoitteesta Nvidia: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/education-projects/

Park, K.-Y.;& Hwang, S.-Y. (2014). Robust Range Estimation with a Monocular Camera for Vision-Based Forward Collision Warning System. *The Scientific World Journal*.

Pham, V.;Nguyen, D.;& Donan, C. (2022). *Road Damages Detection and Classification with YOLOv7.* Huntsville: Computer Science Department Sam Houston State University.

Pomerat, J.;Segev, A.;& Datta, R. (2019). On Neural Network Activation Functions and Optimizers in Relation to Polynomial Regression. *2019 IEEE International Conference on Big Data* (ss. 6183-6185). IEE: Los Angeles.

Pragati, B. (28. March 2023). *Activation Functions in Neural Networks [12 Types & Use Cases]*. Noudettu osoitteesta v7labs: https://www.v7labs.com/blog/neural-networks-activation-functions

Ramsundar, B.;& Zadeh, R. B. (2018). *TensorFlow for Deep Learning.* Sebastopol: O'Reilly Media.

Roh, Y.;Heo, G.;& Whang, S. E. (2021). A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4*, 1328-1347.

Rumelhart, D. E.;Hinton, G. E.;& Williams, R. J. (09. 10 1986). Learning representations by back-propagating errors. *Nature*, 533-536.

Sanida, T.;Sideris, A.;& Dasygenis, M. (2020). A Heterogeneous Implementation of the Sobel Edge Detection Filter Using OpenCL. *2020 9th International Conference on Modern Circuits and Systems Technologies* (ss. 1-4). IEEE: Bremen.

Sazanita Isa, I.;Rosli, M.;Yusof, U.;Maruzuki, M.;& Sulaiman, S. (2022). Optimizing the Hyperparameter Tuning of YOLOv5 for Underwater Detection. *Open Access Journal*.

Serrador, P. (2013). *The Impact of Planning on Project Success-A Literature Review.* Researchgate.

Shah, D. (14. January 2023). *data-augmentation-guide*. Noudettu osoitteesta v7labs: https://www.v7labs.com/blog/data-augmentation-guide

Solving. (23. 10 2022). *Products*. Noudettu osoitteesta Solving: https://www.solving.com/

Sormunen, T. (2023). *Pallet detection in warehouse environment.* Tampere: Aalto University.

Srivastava, N.;Hinton, G.;Krizhevsky, A.;Sutskever, I.;& Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 1929-1958.

Szeliski, R. (2022). *Computer Vision : Algorithms and Applications.* Cham Springer International Publishing AG 2022.

Wang, C.-Y.;Bochkovskiy, A.;& Mark Liao, H.-Y. (2022). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object.* Institute of Information Science, Academia Sinica, Taiwan.

Wang, H.;& Song, Z. (2020). Improved Mosaic: Algorithms for more Complex Images. *Journal of Physics Conference Series 1684*.

Wang, S.;& Su, Z. (2021). Metamorphic Object Insertion for Testing Object Detection Systems. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 1053-1065.

Wood, T. (10. 01 2023). *Softmax Function*. Noudettu osoitteesta DeepAI: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

Xiang, L.;Kaipeng, D.;Guanzhong, W.;Yang, Z.;Qingqing, D.;Gao, Y.;. . . Shilei, W. (2020). *PP-YOLO: An Effective and Efficient Implementation of Object Detector.*

Ximea. (19. 11 2022). *Jetson_Nano_Benchmarks*. Noudettu osoitteesta Ximea: https://www.ximea.com/support/wiki/apis/Jetson_Nano_Benchmarks

Yang, Y.;K.C. Lee, P.;& Cheng, T. (2016). Continuous improvement competence, employee creativity, and new service development performance: A frontline employee perspective. *The International Journal of Production Economics*, 275-288.

Yingjie, T.;Duo, S.;Stanislao, L.;& Xiaohui, L. (2022). Recent advances on loss functions in deep learning for computer vision. *Neurocomputing*, 129-158.

Yorioka, D.;Kang, H.;& Iwamura, K. (2020). Data Augmentation For Deep Learning Using Generative Adversarial Networks. *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)* (ss. 516-518). Kobe: IEEE.

Zhang , X.;Wang, Y.;Zhang , N.;Xu , D.;& Chen, B. (2019). *Research on Scene Classification Method of High-Resolution Remote Sensing Images Based on RFPNet.* Beijing : University of Chinese Academy of Sciences.

Zhao, S.;Louidor, E.;& Gupta, M. (2022). Global Optimization Networks. *Proceedings of the 39th International Conference on Machine Learning*, (ss. 26927-26957).

Zhaohui, Z.;Ping, W.;Wei, L.;Jinze, L.;Rongguang, Y.;& Dongwei, R. (2020). Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *12993-13000*, (ss. 12993-13000).

Zhong-Qiu, Z.;Peng, Z.;Shou-Tao, X.;& Xindong, W. (209). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems* (ss. 3212 - 3232). IEEE.

Appendix 1. List of objects AGVs encounter when operating

Object	Does the current system recognize it?	Is there a risk for the AGV to encounter and crash into it?	Ignored	Trained to recognize	Additional information
AGV	Yes	No	Yes	No	
AGV for a machine assembly line	Yes	No	Yes	No	
Counterbalance forklift	No	Yes	No	Yes	
Dock door	Yes	No	Yes	No	
Hanging large objects above 2m	No	Yes	Yes	No	
Hanging large objects below 2m	Yes	No	Yes	No	
Order Picker	No	Yes	No	Yes	
Other transportation devices	Yes	No	Yes	No	
Pallet	Yes	Yes	Yes	No	
Pallet Jack	No	Yes	No	Yes	
Pallet truck	No	Yes	No	Yes	
Person	Yes	No	Yes	No	
Reach Truck	No	Yes	No	Yes	Does not damage the AGV
Smaller wooden parts	Yes/No	Yes	Yes	No	Does not damage the AGV
Traffic cone	Yes	No	Yes	No	
Trash	Yes/No	Yes	Yes	No	
Trash bin	Yes	No	Yes	No	
Wooden box	Yes	No	Yes	No	Not visible for the camera

Appendix 2. Python commands

```
!python script.py --input LatestDataset.json --names
labelNames.txt --output labels

!python script2.py --input Reach-truck-val.mp4 --output
images --frame-rate 30

!git clone https://github.com/WongKinYiu/yolov7

!pip install -r requirements.txt

!pip3 install opencv-python-headless==4.5.3.56

!wget
https://github.com/WongKinYiu/yolov7/releases/download/v0.1
/yolov7_training.pt

import os
from sklearn.model_selection import train_test_split
path1 = "./images-train/"
path2 = "./images-validation/"
path3 = "./images-test/"

image_names1 = os.listdir(path1)
image_names2 = os.listdir(path2)
image_names3 = os.listdir(path3)

train_set = os.listdir(path1)
val_set = os.listdir(path2)
test_set = os.listdir(path3)


from time import time

def print_time(t: float):
    print(f'Took {time()-t:.2f} s')

def train_val_test_split(image_names: list[str], sample_rate:
int, val_perc: int, test_perc: int) -> tuple[str, str, str]:
    print(f'Splitting to train, val, test with sample rate
of {sample_rate}')
    t = time()
    train_names,                    val_names                    =
train_test_split(image_names[::sample_rate],
test_size=(val_perc + test_perc) / 100, shuffle=False)
    val_names,  test_names  =  train_test_split(val_names,
test_size=(test_perc)    /    (val_perc    +    test_perc),
shuffle=False)
    print_time(t)
    return train_names, val_names, test_names
```

```
def save_image_names1(names: list[str], filename: str,
folder_path) -> None:
    file_path = f'{folder_path}/{filename}'
    print(f'Saving image names to {file_path}')
    t = time()
    names_str = ''
    for name in names:
        names_str += IMAGES_PATH + name + '\n'
    os.makedirs(folder_path, exist_ok=True)
    with open(file_path, 'w') as f:
        f.write(names_str)
    print_time(t)
```

```
IMAGES_PATH = './images/'
SPLITS_PATH = './splits/'

SAMPLE_RATE = 1

image_names_folder_path =
f'{SPLITS_PATH}/sample_{SAMPLE_RATE}'

save_image_names1(train_set, 'train.txt',
image_names_folder_path)
save_image_names1(val_set, 'val.txt',
image_names_folder_path)
save_image_names1(test_set, 'test.txt',
image_names_folder_path)
```

```
!python train.py --batch 16 --epochs 30 --data data/data.yaml
--weights './runs/train/exp107/weights/last.pt' --device 0

!python test.py --
weights ./runs/train/exp37/weights/last.pt --task test --
data data/data.yaml --batch 16 --device 0

!python detect.py --
weights ./runs/train/exp108/weights/last.pt --conf 0.9 --
source FalsePredTest.mp4
```

Appendix 3. Hyperparameters of the tenth model

```
lr0: 0.001  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.01  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0   # focal loss gamma (efficientDet default
gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.2  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range
0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 0.5  # image mosaic (probability)
mixup: 0.1  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.05  # image copy paste (probability), use 0 for
faster training
loss_ota: 1 # use ComputeLossOTA, use 0 for faster training
```

Appendix 4. Training results

Figure 15 Results of the third training run.

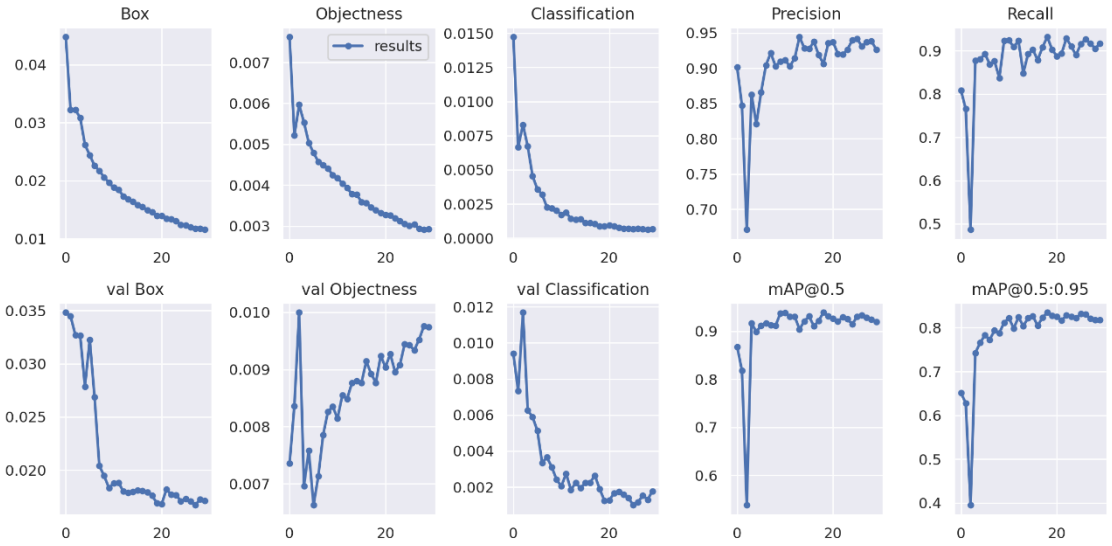


Figure 16 Results of the fifth training run with a larger dataset.

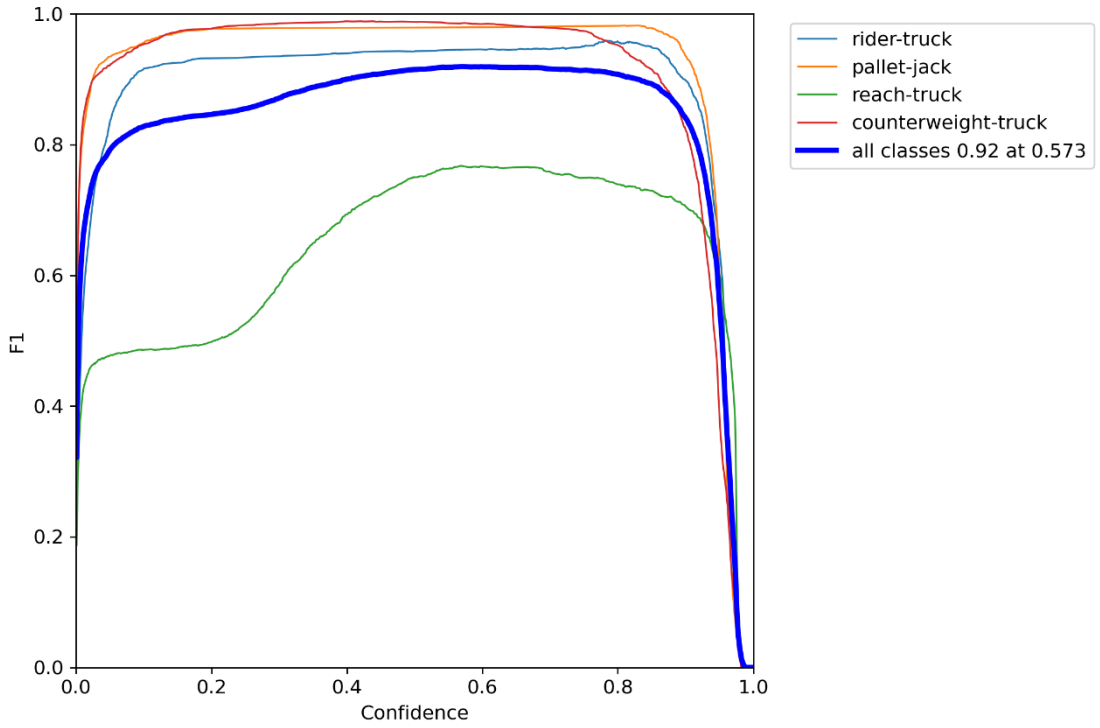


Figure 17 F1 score of the fifth training run.

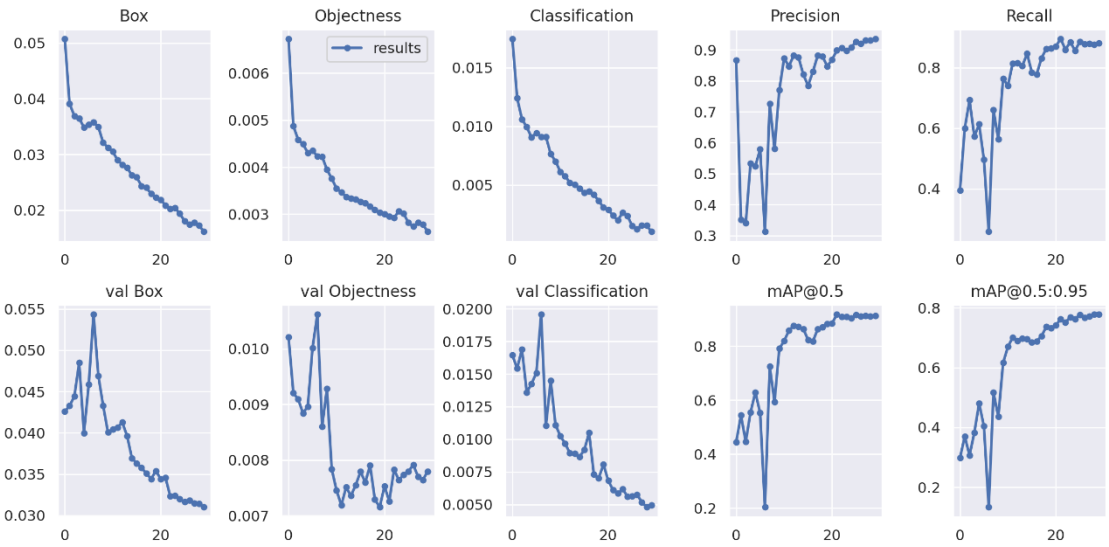


Figure 18 Results of seventh training run with lower mosaic hyperparameter.

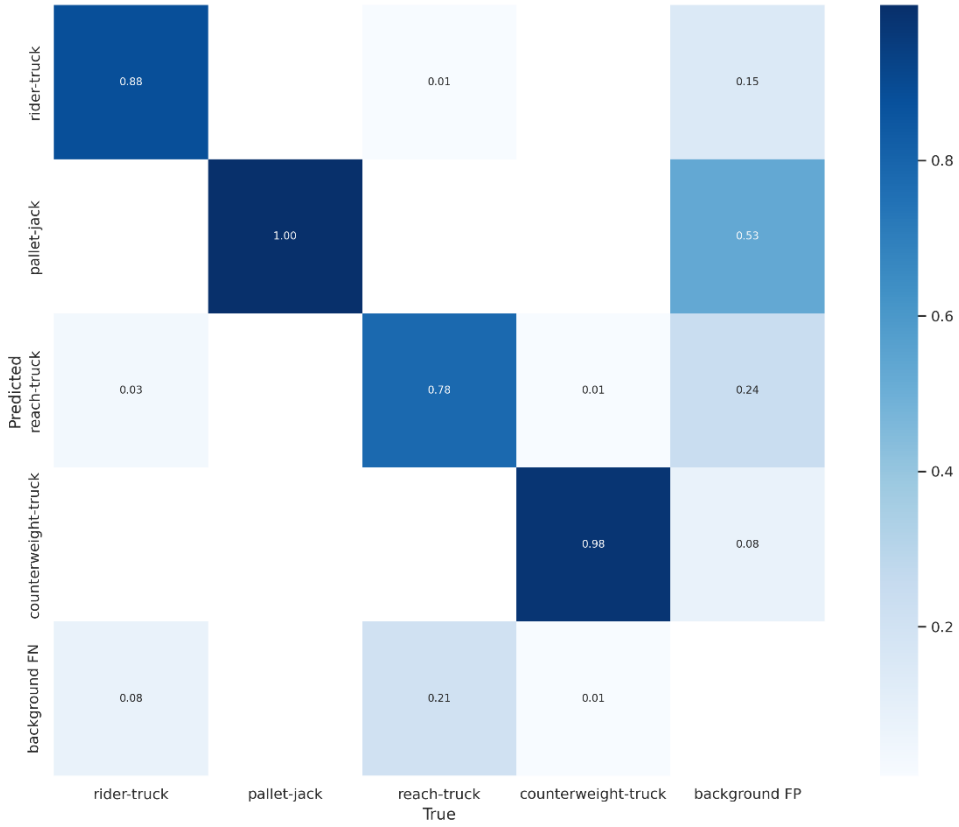


Figure 19 Confusion matrix after the seventh training run.

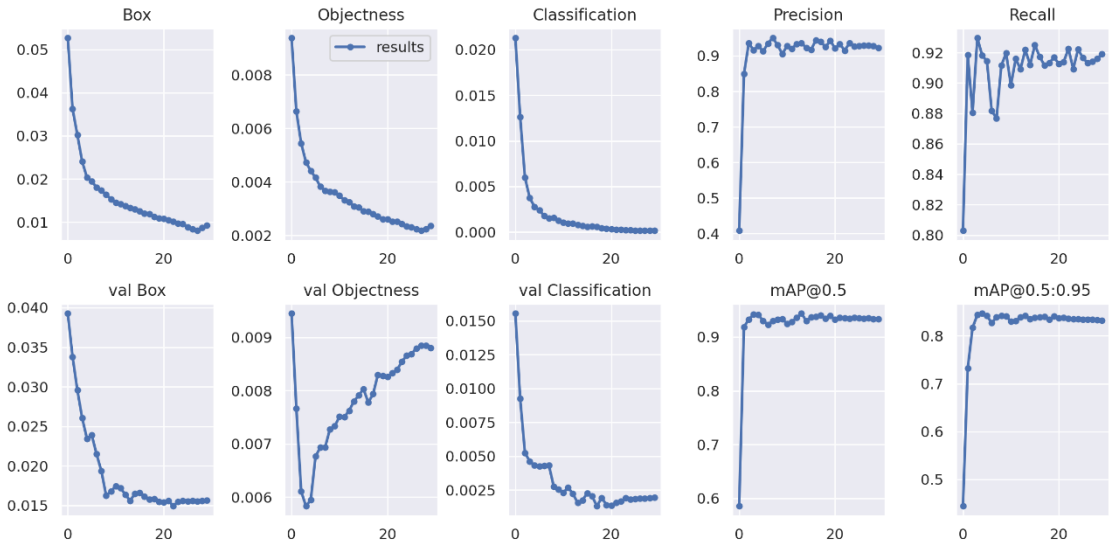


Figure 20 Results of the ninth training run with lower learning rate and initial mosaic parameter.

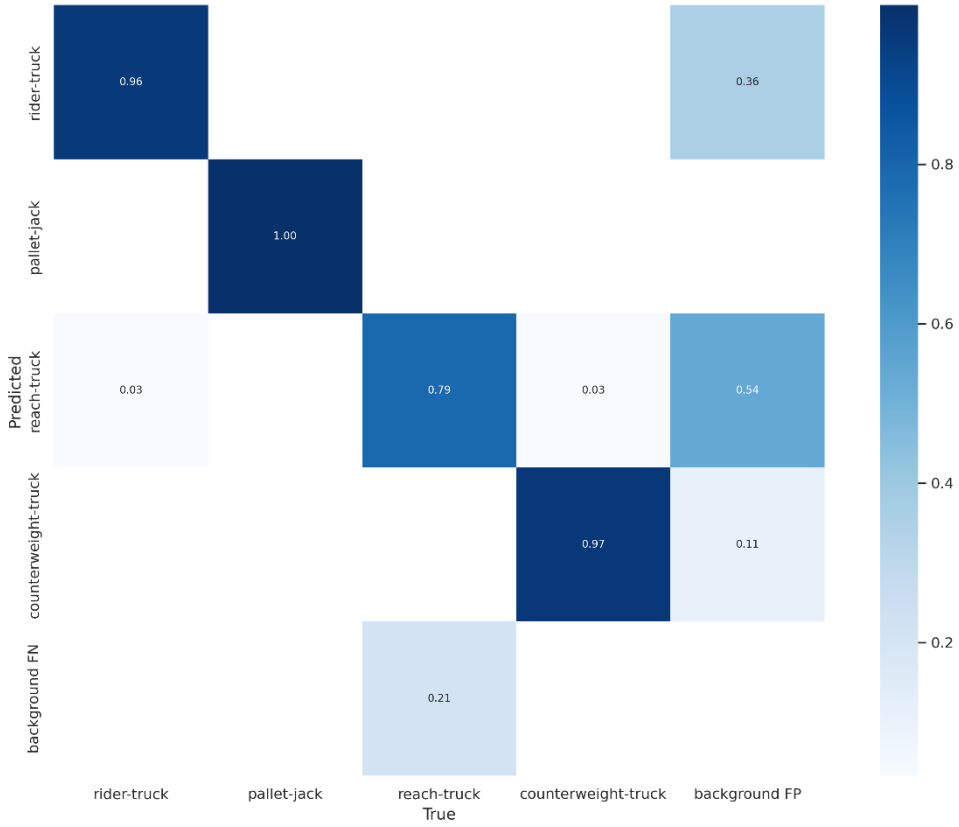


Figure 21 Confusion matrix of the ninth training run.

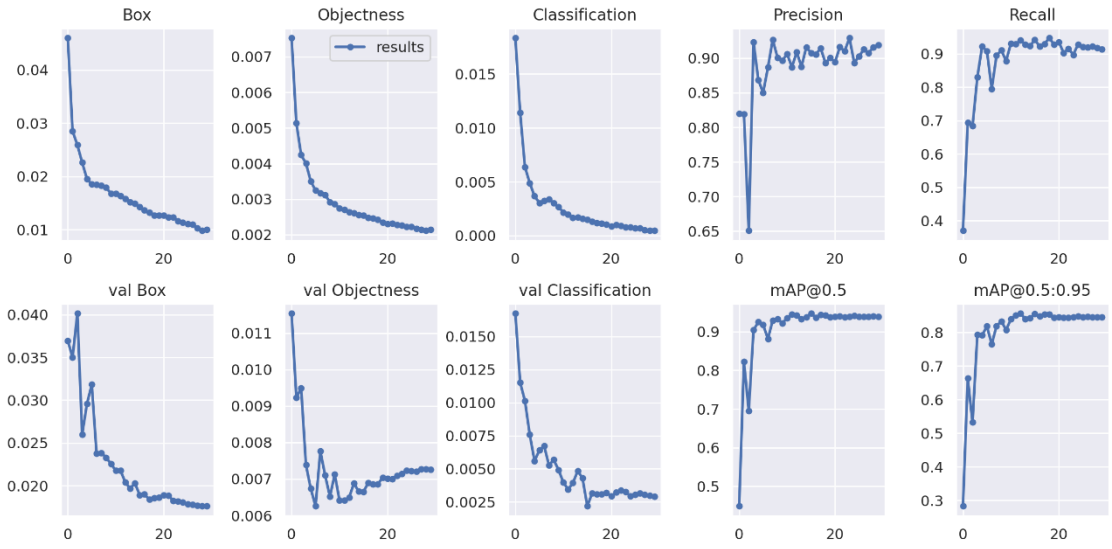


Figure 22 Results of the tenth training run with adjusted augmentation parameters.

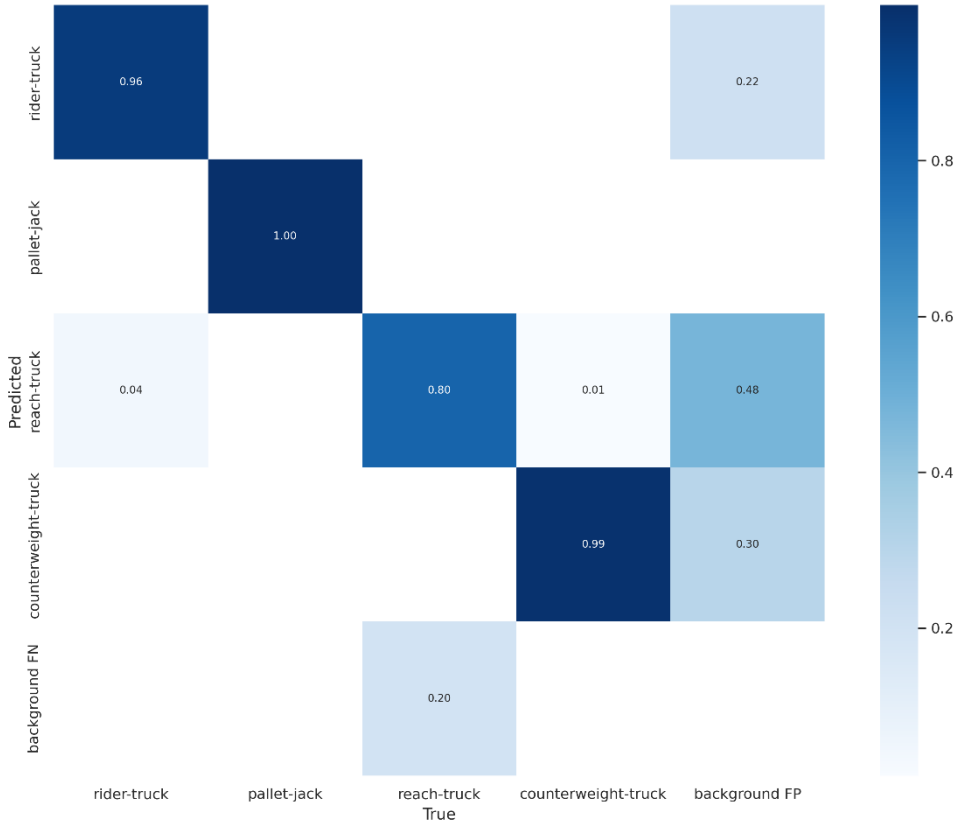
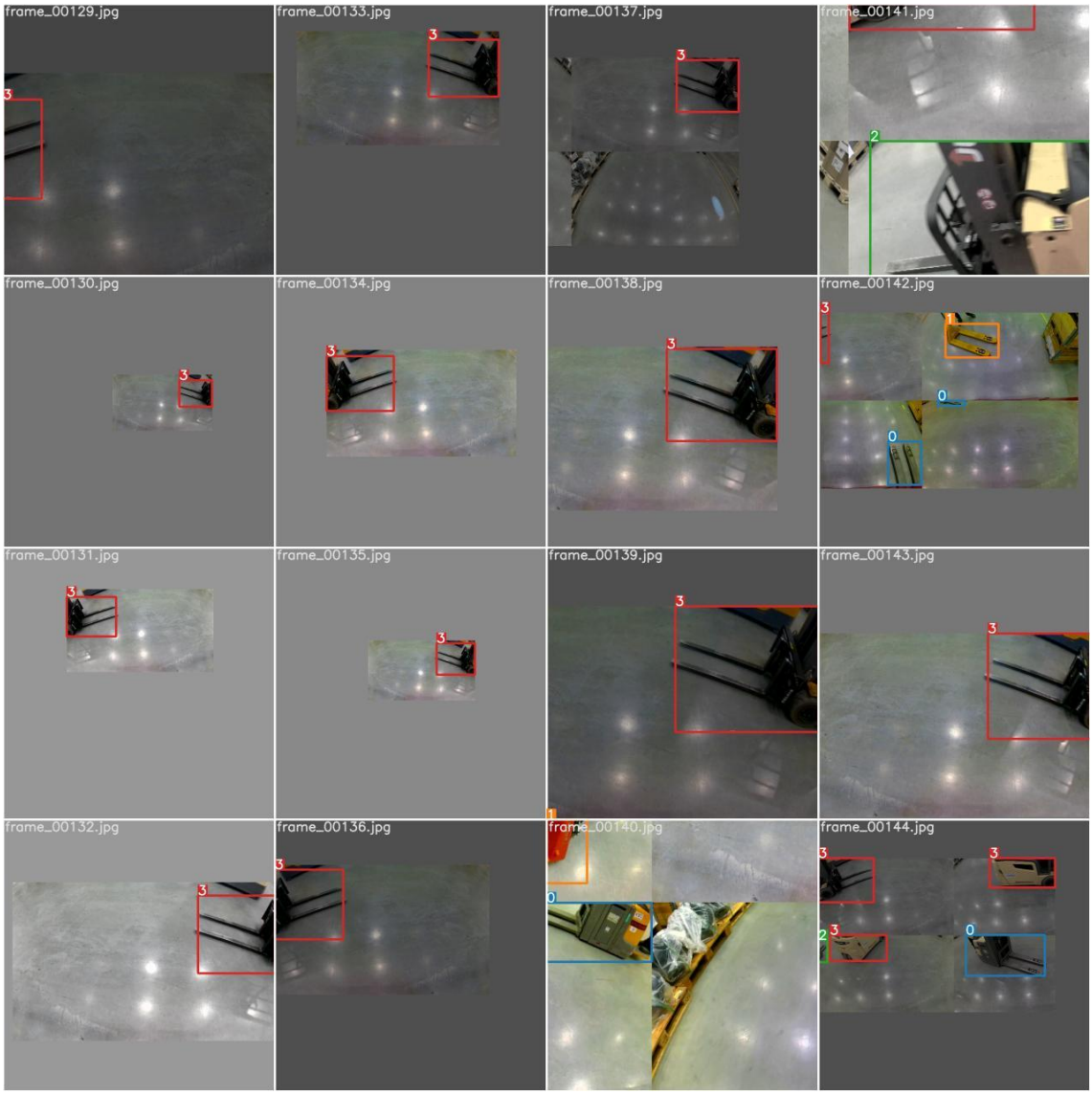


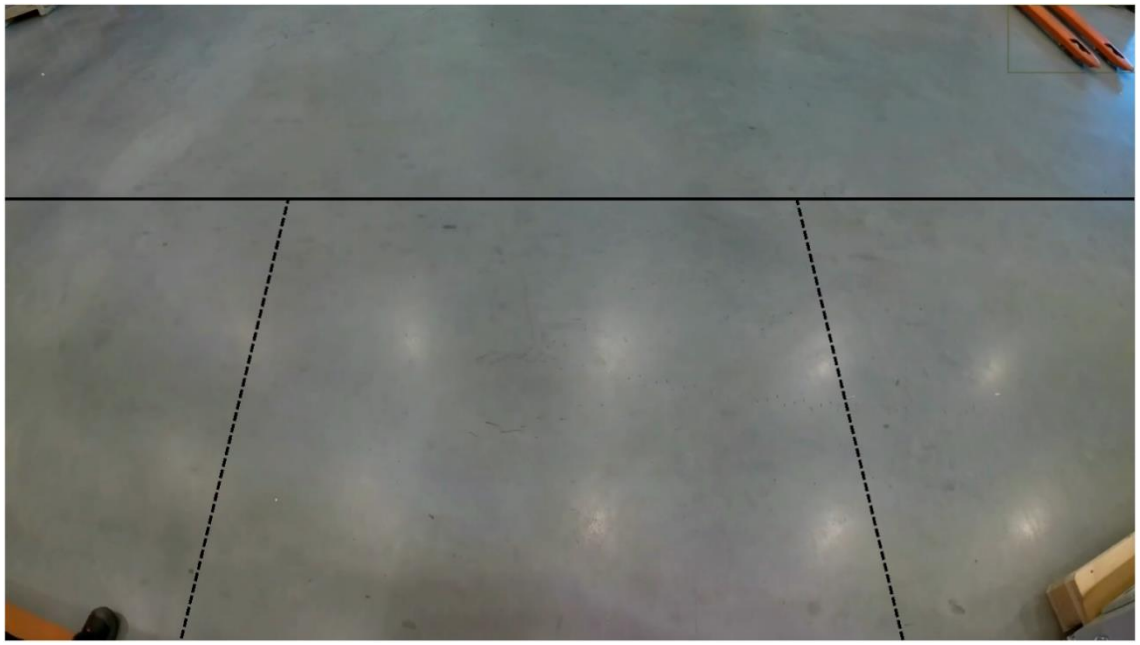
Figure 23 Confusion matrix of the tenth training run.



Picture 24 Example of a training batch used to train model number ten.

Appendix 5. Test cases

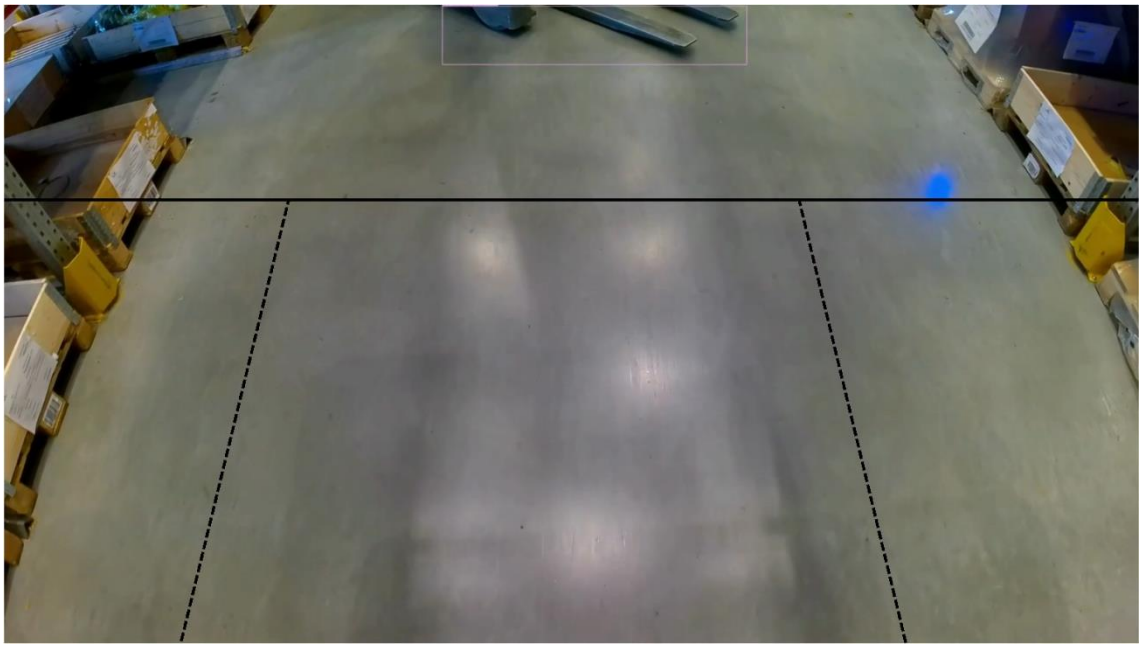
Picture 25 Test case 1 for pallet-jack.



Picture 26 Test case 2 for pallet-jack.

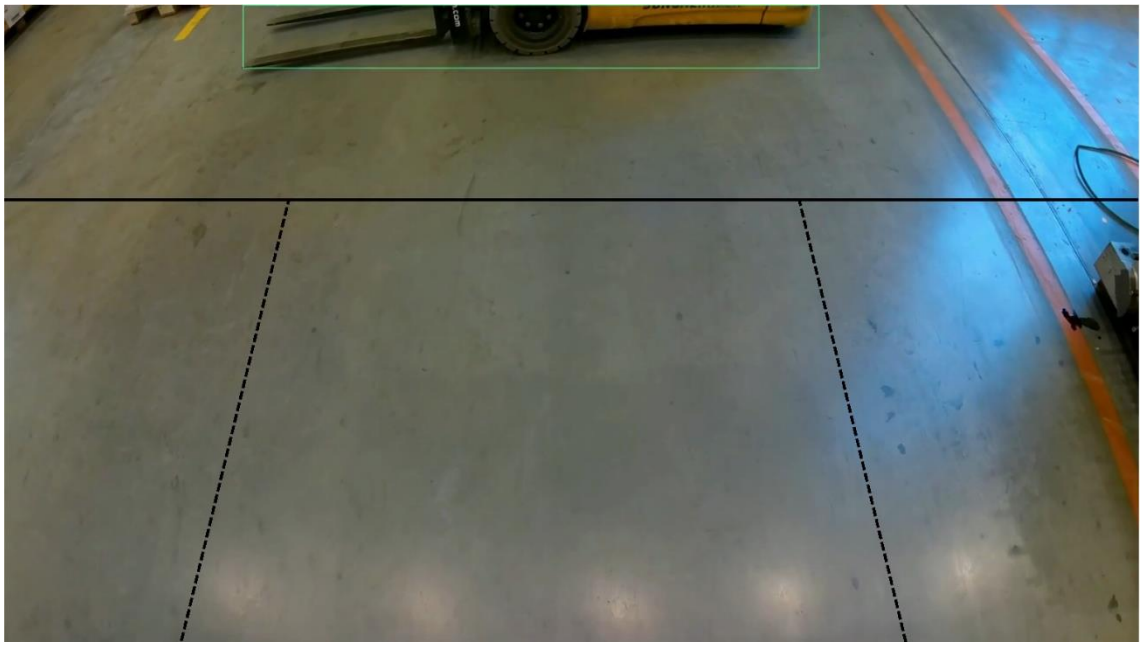
Picture 27 Test case 1 for rider-truck.

Picture 28 Test-case 2 for rider-truck.

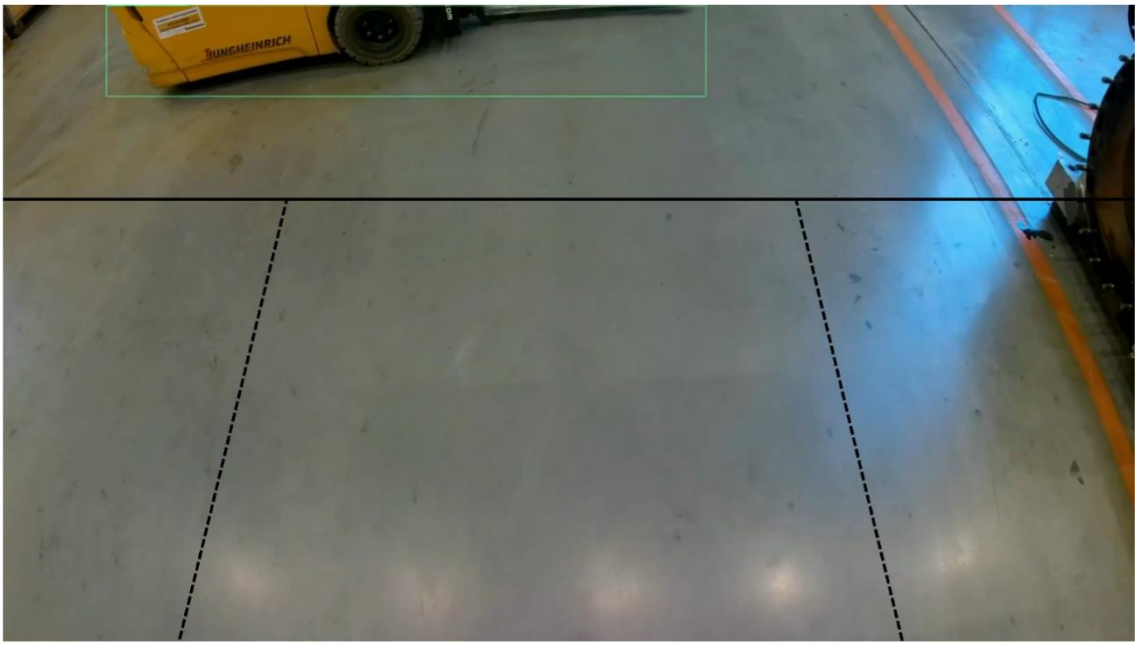


Picture 29 Test case 1 for reach-truck.

Picture 30 Test case 2 for reach-truck.



Picture 31 Test case 1 for counterweight-truck.



Picture 32 Test case 2 for counterweight-truck.

Appendix 6. Dependencies

```
# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5,<1.24.0
opencv-python>=4.1.1
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch==1.7.1,!=1.12.0
torchvision>=0.8.1,!=0.13.0
tqdm>=4.41.0
protobuf<4.21.3

# Logging -------------------------------------
tensorboard>=2.4.1
# wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
# coremltools>=4.1  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.3.6  # ONNX simplifier
# scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export

# Extras --------------------------------------
ipython  # interactive notebook
psutil  # system utilization
thop  # FLOPs computation
# albumentations>=1.0.3
# pycocotools>=2.0  # COCO mAP
# roboflow
# opencv-python-headless==4.5.3.56
```