



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Moaz Elshebly

Autonomous Garbage Detection and Localization using Deep Learning and Computer Vision Techniques

*Fakultät Technik und Informatik
Studiendepartment Informations
und Elektrotechnik*

*Faculty of Engineering and Computer Science
Department of Information and Electrical
Engineering*

Moaz Elshebly

**Autonomous Garbage Detection and Localization
using Deep Learning and Computer Vision
Techniques**

Bachelor Thesis based on the examination and study regulations for the Bachelor of
Science Information Engineering

at the Department Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the Hamburg University of Applied Sciences

Supervising Examiner: Prof. Dr. Jörg Dahlkemper
Second Examiner: Prof. Dr. Marc Hensel

Delivered on: 6. October 2020

Moaz Elshebly

Title of the paper

Autonomous Garbage Detection and Localization using Deep Learning and Computer Vision Techniques

Keywords

Machine Learning, Deep Learning, Real-time Object Detection, Convolutional Neural Networks, YOLO, TensorFlow, OpenCV, Keras, Computer Vision, Image Processing, Python

Abstract

Environmental pollution is a serious issue that the world is facing nowadays. A tremendous amount of waste is generated every year causing irreparable damage and permanent pollution to the Earth. Moreover, waste generated by humans in public places shows an uncivilized image of some nations which is why it is very important to rely on modern solutions to get rid of such issues. This thesis aims to utilize advanced deep learning and computer vision techniques in order to detect waste in public places and help getting rid of such waste in an automated way.

Within the scope of this thesis, different approaches of object detection using pre-trained deep neural networks are discussed. Furthermore, it also comprises the pre-processing pipeline of collecting potential input data, preparing it and applying the concept of data augmentation on it.

The final assessment includes current results and further development and application of the aforementioned object detection model.

Thema der Arbeit

Autonomous Garbage Detection and Localization using Deep Learning and Computer Vision Techniques

Stichworte

Maschinelles Lernen, Deep Learning, Echtzeit-Objekterkennung, Faltungsnetze, YOLO, TensorFlow, OpenCV, Keras, Computer Vision, Bildverarbeitung, Python

Kurzzusammenfassung

Die Umweltverschmutzung ist ein ernstes Problem, mit dem die Welt heutzutage konfrontiert ist. Jedes Jahr fallen enorme Abfallmengen an, die irreparable Schäden und eine dauerhafte Verschmutzung der Erde verursachen. Darüber hinaus ist der von Menschen an öffentlichen Orten erzeugte Abfall ein unzivilisiertes Bild einiger Nationen, weshalb es sehr wichtig ist, sich auf moderne Lösungen zu verlassen, um solche Probleme zu lösen. Diese Thesis zielt darauf ab, fortgeschrittene Techniken des Deep Learning und der rechnergestützten Bildverarbeitung zu nutzen, um Abfall an öffentlichen Plätzen aufzuspüren und zu helfen, solchen Abfall auf automatisierte Weise entsorgen.

Im Rahmen dieser Arbeit werden verschiedene Ansätze der Objekterkennung mit Hilfe von vortrainierten tiefen neuronalen Netzen diskutiert. Darüber hinaus umfasst sie auch die Vorverarbeitungspipeline der Sammlung potenzieller Eingabedaten, deren Aufbereitung und die Anwendung des Konzepts der Data Augmentation auf diese Daten.

Die abschließende Bewertung umfasst die aktuellen Ergebnisse sowie die Weiterentwicklung und Anwendung des oben genannten Objektdetektionsmodells.

Contents

1	Introduction	1
1.1	Problem Statement	1
1.2	Objective	2
1.3	Document Structure	3
2	State of the Art	4
2.1	Related Work	4
2.2	Artificial Intelligence	5
2.3	Object Detection and Computer Vision	8
2.3.1	Convolutional Neural Networks	9
2.3.2	Fully Convolutional Networks	11
2.3.3	Region-based Convolutional Neural Networks	11
2.3.4	Fast-er R-CNNs	12
2.3.5	Single Shot Detectors	14
2.3.6	You Only Look Once	15
3	Requirements Analysis	20
3.1	Overview	20
3.2	Functional Requirements	22
3.3	Non-functional Requirements	23
4	Conception	26
4.1	Network Architecture	26
4.2	Hardware	28
4.2.1	Training Hardware	28
4.2.2	Detection Hardware	29
4.3	Software Technologies	29
4.3.1	Programming Language	29
4.3.2	Machine Learning Framework	30
4.3.3	Computer Vision Library	32

5	Development	35
5.1	Dataset	35
5.1.1	Data Collection	36
5.1.2	Data Pre-processing	39
5.2	Model Implementation	42
6	Evaluation	47
6.1	Evaluation Metrics	47
6.2	Experimentation and Results	49
7	Summary	59
7.1	Conclusion	59
7.2	Future Work	60

List of Figures

1.1	Generated waste per person in the EU in 2005 versus in 2018 [3]. . . .	2
2.1	Artificial intelligence, machine learning, and deep learning [15]. . . .	6
2.2	Machine learning vs classical programming [15].	7
2.3	Typical structure of a Deep Neural Network.	8
2.4	CNN layers with rectangular local receptive fields [24].	10
2.5	R-CNN: Regions with CNN features [27].	11
2.6	Architecture of a Fast R-CNN [28].	12
2.7	Architecture of a Faster R-CNN with the RPN [29].	13
2.8	Comparison in hours between ordinary, Fast, and Faster R-CNNs trained on the same set of data[30].	13
2.9	Architecture of a SSD object detection model [31].	14
2.10	The YOLO object detection system at a glance [32].	15
2.11	Architecture of a YOLO object detection model [32].	16
2.12	Speed/Accuracy trade-off of different state-of-the-art object detection systems [36].	17
2.13	Comparison between YOLOv4 and other state-of-the-art object detectors [37].	18
2.14	Comparison between YOLOv5 and EfficientDet [39].	19
3.1	The life-cycle of a machine learning project [40].	21
3.2	Data preparation process [41].	21
5.1	Samples of images from the trashnet dataset [50].	37
5.2	Samples of collected images from the Let's Do It Foundation dataset [51].	38
5.3	An example of augmented images (right) in comparison to original images (left).	40
5.4	Some of the popular methods used for image annotation [54].	41
5.5	Flowchart of the training implementation.	45
6.1	Precision vs Recall curve [24].	48
6.2	Training vs Validation losses during first training iteration.	50
6.3	Training vs Validation losses during second training iteration.	51

List of Figures

6.4	MAE values during second training iteration.	51
6.5	Training vs Validation losses during final training iteration.	53
6.6	MAE values during final training iteration.	53
6.7	AP@0.5 of the final model.	54
6.8	mAP of various object detection networks trained on COCO dataset [58].	54
6.9	Model predictions on some images of the test dataset.	55
6.10	Training vs Validation losses for Tiny YOLO architecture.	56
6.11	AP@0.5 of the tiny model.	57
6.12	Tiny model predictions on some images of the test dataset.	58

List of Tables

3.1	Summary of the project's functional requirements.	23
3.2	Popular CNN-based architectures used in computer vision applications and the sizes of the datasets used for their training [42].	24
3.3	Summary of the project's non-functional requirements.	25
4.1	Comparison of available state-of-the-art object detection architectures.	27
4.2	Comparison of the most popular ML and DL frameworks.	32
4.3	Comparison of the most popular computer vision libraries.	34

Listings

5.1	Unfreezing all the layers of the model.	44
5.2	Reduce learning rate callback.	44
6.1	Early stopping callback when the bottom layers are frozen.	52
6.2	Early stopping callback when the bottom layers are unfrozen.	52

Acronyms

AI Artificial Intelligence. 4–6, 29, 31

AP Average Precision. viii, 17, 18, 47–49, 54, 57

API Application Programming Interface. 30, 32, 33

CNN Convolutional Neural Network. vii, 9–12, 15

DL Deep Learning. ix, 5, 6, 30–32

DNN Deep Neural Network. 10, 43

FCN Fully Convolutional Network. 11, 12

FPS Frames per Second. 15, 17, 23, 29, 49, 54, 55, 57

GPU Graphical Processing Unit. 17, 23, 25, 28, 29, 31, 32, 54, 57

IDE Integrated Development Environment. 33

IoU Intersection over Union. 49

MAE Mean Absolute Error. viii, 47, 49–53

ML Machine Learning. ix, 5, 6, 11, 25, 28–32, 34, 41

R-CNN Region-based Convolutional Neural Network. vii, 4, 11–14, 16, 27

RPN Region Proposal Network. vii, 12–14

SSD Single Shot Detector. vii, 14, 16, 27

SVM Support Vector Machine. 11

Acronyms

VM Virtual Machine. 28

YOLO You Only Look Once. iii, iv, vii, viii, 5, 15–19, 26–28, 33, 42, 54, 56, 57, 60

1 Introduction

1.1 Problem Statement

The waste humans generate has been detrimental to the environment for quite some time now. Humans are generating tremendous amounts of trash that cannot be dealt with in a sustainable way. Waste that is not biodegradable and cannot be properly be recycled is filling the oceans and landfills. A recent study about plastic waste found that of the 6.3 billion metric tons of plastic waste that has been produced [1], only 9% of that plastic waste had been recycled and only 10% of which had been recycled more than once.

Poor waste management contributes to climate change, air pollution, and directly affects many ecosystems and species. Some ecosystems, like the marine and coastal ones, can be severely affected by poor management of waste. Waste impacts the environment indirectly as well; whatever is not recycled or recovered from it represents a loss of raw material and other inputs used in the chain, i.e. in the production, transport, and consumption phases of the product. Environmental impacts in the life-cycle chain are significantly larger than those in the waste management phases alone [2].

Waste also represents an economic loss and burden to the society; labor and the other inputs (land, energy, etc.) used in its extraction, production, dissemination, and consumption phases are also lost when the 'leftovers' are discarded. Moreover, waste management costs money; creating an infrastructure for collecting, sorting and recycling is costly, but once in place, recycling can generate revenues and create jobs.

1.2 Objective

A statistical report in 2018 showed that the estimated average waste generated per person per year in the European Union is 492 kg. According to the report, and as shown in Figure 1.1, Germany comes in the fourth place in the EU countries with respect to waste generated per person. It can also be seen that the amount of waste generated per person per year in Germany has gone from 433 kg in 2005 to 615 kg in 2018 [3].

Even though it has become obvious that producing such amounts of waste every year is inevitable, governments are trying to reduce the negative effects of the produced waste on the environment by following some techniques such as waste recycling. And since modern problems require modern solutions, science and technology have a substantial role to play in helping getting rid of waste in more effective and environment-friendly ways.

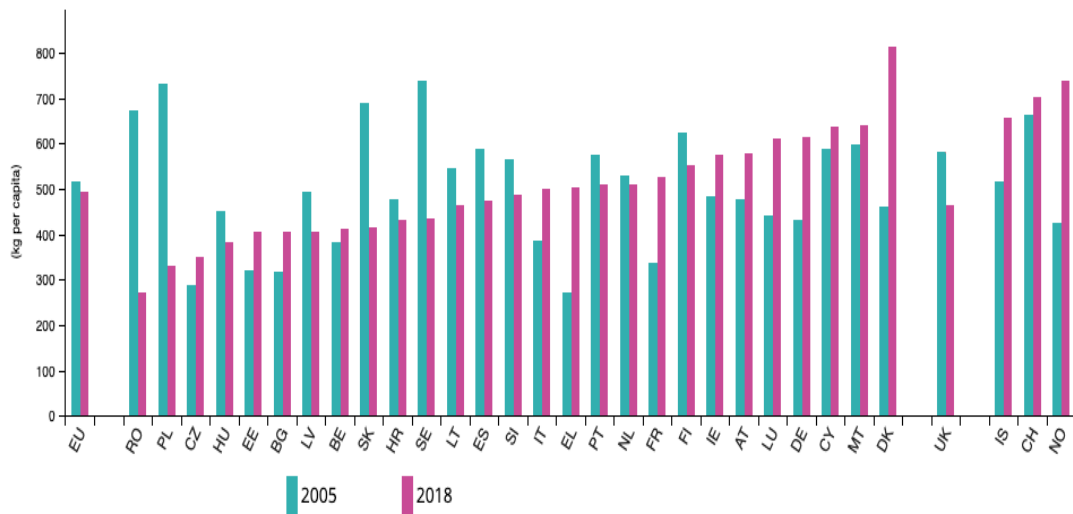


Figure 1.1: Generated waste per person in the EU in 2005 versus in 2018 [3].

The goal of this bachelor thesis is to develop a machine learning model that can be deployed to a drone camera and which is able to detect waste in public places such as parks and streets in real-time. For that purpose, several deep neural networks for object detection and computer vision are to be examined to conclude which one fits

better for the requirements. Such a model would help countries be able to get rid of public waste in an autonomous way in the future by being capable of detecting trash in public places and identifying their exact location.

1.3 Document Structure

This document consist of 7 chapters and is structured as follows: chapter 2 gives and overview of the existing computer vision technologies and state-of-the-art object detection methodologies.

In chapter 3, analysis of the project requirements is performed and the most important features of the project are extracted. Chapter 4 then proceeds to explain how the concluded requirements are met and which technologies are to be used in order to achieve the required results.

Chapter 5 contains all the technical details of the data preparation and implementation of the project. In chapter 6, a thorough evaluation of the results is done and the project's outcome is discussed. Finally, chapter 7 gives a brief summary of this document and ideas for further improvements that can be done to the project.

2 State of the Art

A waste disposal system is a crucial sector in human civilization and the world is currently suffering due to the irresponsible disposal of trash. Nowadays, the workers assigned to collect and dispose of trash live an inhumane life especially in developing countries and they also live at the risk of health hazards and infectious diseases [4]. A smart waste collection system can free those people from such a burden and offer huge advantages to the whole society.

2.1 Related Work

Researchers have been doing significant efforts to come up with smart garbage detection and collection systems, however, there has not been too much attention given to such projects. For instance, a UNESCO prizewinner from Estonia, the [Let's Do It Foundation](#), had the idea to create an image-based trash detection system using Artificial Intelligence (AI) [5]. Their plan was to collect images of trash from various places around the globe, prepare the images to be fed into a Mask Region-based Convolutional Neural Network (R-CNN) to train it for trash detection and finally validate and test the results produced by the model to ensure its accuracy and consistency. Despite the fact that their research was fruitful and they were able to design such a model, they did not receive much attention neither did they have enough resources to continue working on their project by achieving an autonomous trash disposal system. Unfortunately, the project has now stopped, and on their official website, they offer to do a project handover for those interested in completing their work[6].

Another project by [Mangusta Technology](#) in the Netherlands involved deploying a model on a camera on an e-Bike customized for the specific purpose of trash detection in the city of Amsterdam [7]. For their model, they used a modified version of Darknet¹ to train a You Only Look Once (YOLO) [9] object detection system. The bike that is equipped with the camera tours around the city marking places with trash in them and sending them to a Google Firestore database². Additionally, they developed an app that adds pinpoints of trash places on the map. Their idea was to give garbage operators the capability of monitoring garbage statistics throughout the city in real-time and from their offices.

Additionally, A plethora of research papers has been carried out to try to find a solution to this problem by using different approaches. In Bangladesh, a study by Hossain et al. implemented a deep learning algorithm deployed on a low-cost robot with a Raspberry Pi and a camera to create an autonomous trash collector [11]. For developing the model they used Keras³ and OpenCV⁴ for object detection. No pre-trained model was used, instead, the model was created from scratch. The model achieved an accuracy of 96% on the test data. Another study by Liu et al. in Shanghai used the YOLOv2 model as a pre-trained network for research on developing a garbage detection system and achieved an accuracy of 89.2% on the test data [14]. Although those results are promising and may look reliable, they have not yet been seen to be adopted by any of the countries they were developed in.

2.2 Artificial Intelligence

In the past few years, AI has been a subject of intense media hype. Machine Learning (ML), Deep Learning (DL), and AI come up in countless articles, often outside of technology-minded publications. A future of intelligent chatbots, self-driving cars, and virtual assistants is promised —a future sometimes painted in a grim light and

¹Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation [8].

²The Firebase Realtime Database is a cloud-hosted database [10].

³Keras is an open-source neural-network library written in Python [12].

⁴OpenCV is a library of programming functions mainly aimed at real-time computer vision [13].

other times as utopian, where human jobs will be scarce, and most economic activity will be handled by robots or AI agents [15].

First, the difference between AI, ML, and DL and how they are related to each other need to be clearly defined. As shown in Figure 2.1, AI is a general field of automating machine behavior which encompasses ML and DL. A concise definition of AI would be "The effort to automate intellectual tasks normally performed by humans", however, AI does not have to involve any learning. For instance, early chess programs only involved hard-coded rules crafted by programmers, which is known as symbolic AI [15].

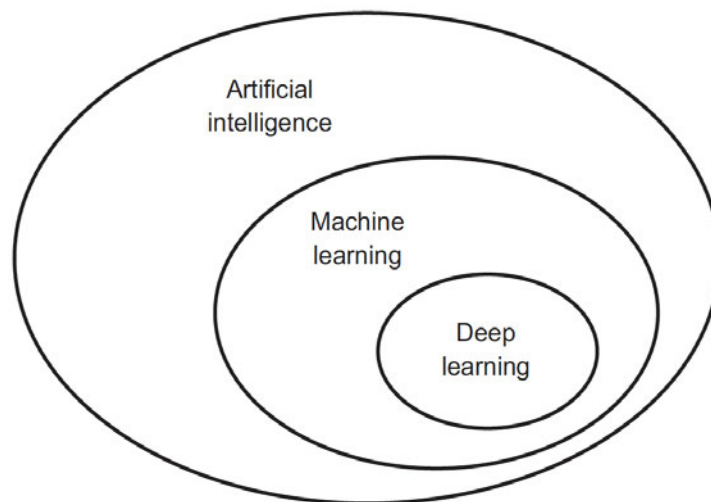


Figure 2.1: Artificial intelligence, machine learning, and deep learning [15].

Machine learning on the other hand is the science of getting computers to act without being explicitly programmed. It is considered as a set of rules that computers use to make and improve predictions or behaviors based on previous data. As it can be seen in Figure 2.2, the system is given a set of previous data relevant to the task at hand alongside the correct results corresponding to this data as inputs. Based on those inputs, the system then tries to define a set of rules that can be applied to similar problems occurring in the future [16].

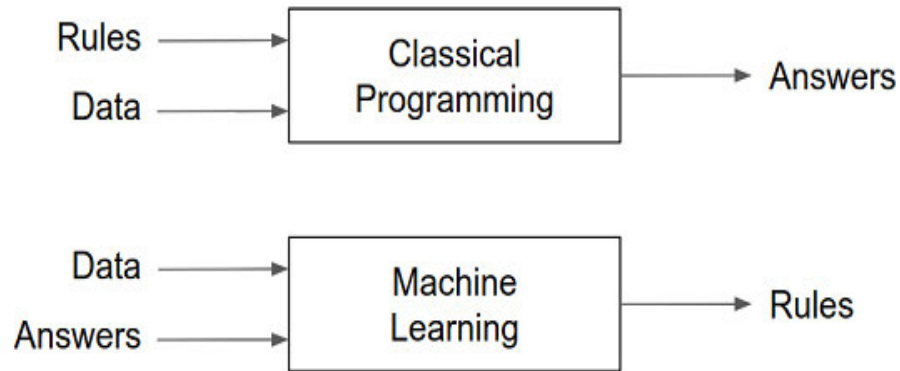


Figure 2.2: Machine learning vs classical programming [15].

This set of input data can be in the form of digitized human-labeled training sets or other types of information obtained via interaction with the environment. In all cases, its quality and size are crucial to the success of the predictions made by the algorithm. A machine-learning model transforms its input data into meaningful outputs by finding an appropriate representation for the input data and make it more amenable to the task at hand. This allows for solving a remarkably broad range of intellectual tasks, from speech recognition to autonomous car driving [17].

A specific subfield of machine learning is deep learning; putting an emphasis on learning consecutive layers by introducing representations that are expressed in terms of other, simpler representations [15][18]. Modern deep learning often involves tens or even hundreds of successive layers of representations— and they’re all taught automatically from exposure to training data. Those layered representations are almost always learned via models called neural networks, structured in literal layers stacked on top of each other. Neural networks are designed to accomplish one small task with high efficiency. They usually consist of an input layer, an output layer, and sometimes hidden layers [19]. A typical structure of a neural network can be seen in Figure 2.3.

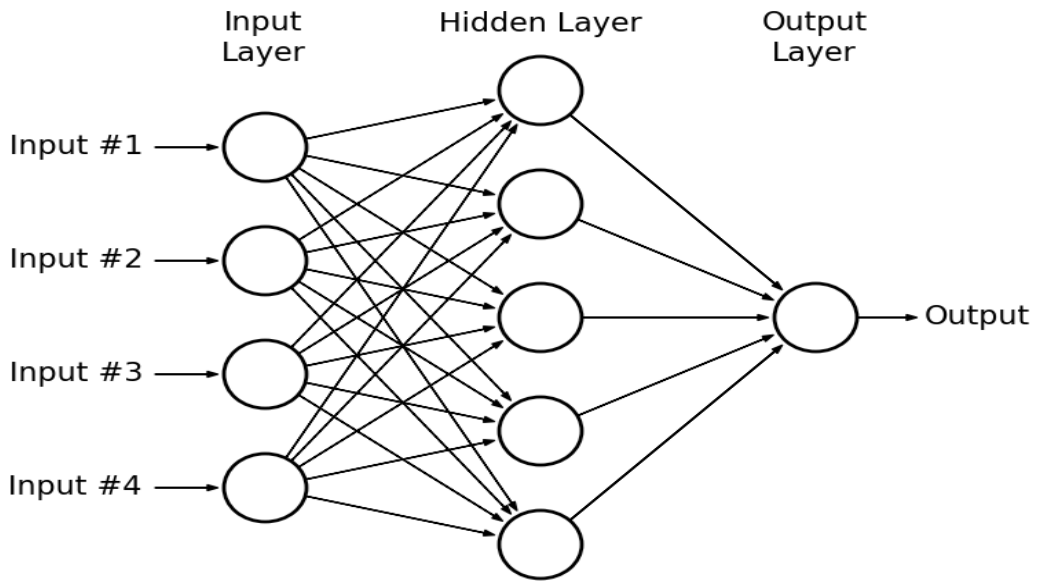


Figure 2.3: Typical structure of a Deep Neural Network.

2.3 Object Detection and Computer Vision

Computer vision is the automated extraction of information from images and videos. Information can mean anything from 3D models, camera position, object detection and recognition to grouping and searching image content [20]. This has proved to be a surprisingly challenging task; it has occupied thousands of intelligent and creative minds over the last decades, and despite this, being able to build a general-purpose “seeing machine” is still a far destination [21].

Object detection is an important research area in image processing and computer vision. It is the task of classifying and localizing multiple objects in an image or a video [22]. Interpreting the object localization can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object.

The performance of object detection systems has significantly improved over the past decade by applying deep learning methodologies. Throughout the following sections, some of those systems and methodologies are discussed in a more detailed manner to provide an overview of the relevant available architectures.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special kind of neural networks for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels [18]. Using such architecture makes CNNs fast to train and this, in turn, helps in training deep, multi-layer networks that are very good at classifying images [23]. The name “Convolutional” Neural Network indicates that the network employs a mathematical operation called convolution⁵. Hence, it can be said that they are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

The most important building block of a CNN is the convolutional layer which performs the aforementioned mathematical convolution operation. Neurons in the first convolutional layer are not densely connected to every single neuron in the input layer as we’ve seen in the classic neural network structure in Figure 2.3, but only connected to pixels that represent their respective field as shown in Figure 2.4. In turn, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer.

This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on. This hierarchical structure is common in real-world images, which is why CNNs have been tremendously successful nowadays in practical applications for image recognition.

⁵Convolution is a mathematical operation on two functions that produces a third function expressing how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it.

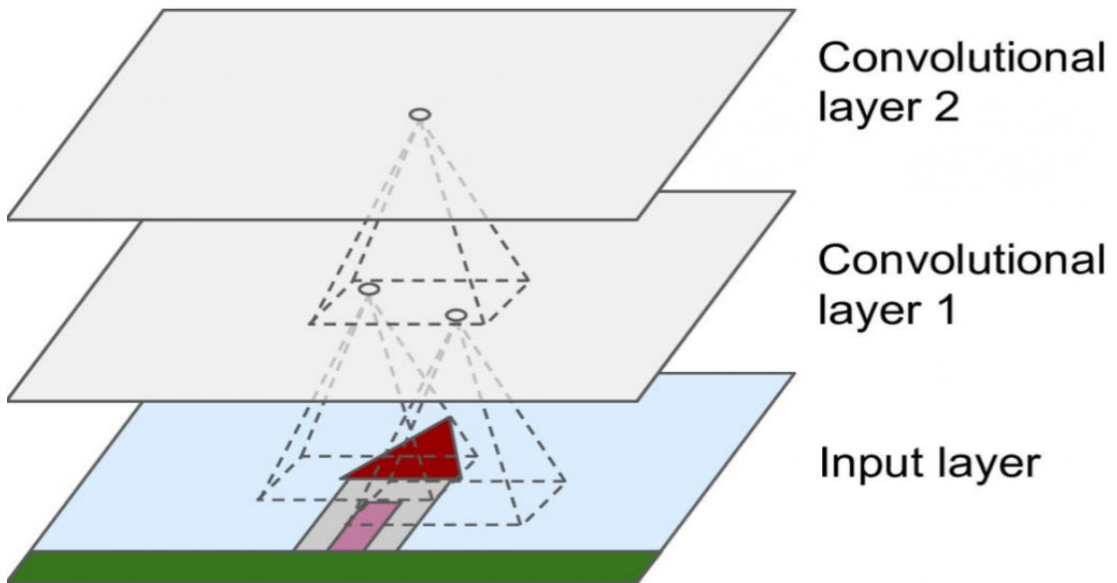


Figure 2.4: CNN layers with rectangular local receptive fields [24].

One of the biggest advantages of CNNs is that once it has learned to recognize a pattern in one location it can recognize it in any other location in the image. On the other hand, once a regular Deep Neural Network (DNN) has learned to recognize a pattern in one location, it can only recognize it in that particular location. This is due to the fact that neurons in each of the CNN layers are responsible for a so-called filter: that is a specific characteristic of the image which is determined by the receptive field.

These so-called filters then output a feature map, which highlights the areas in an image that activate the filter the most. Those filters do not have to be manually defined, rather, the most useful ones for the task at hand are learned automatically during training through the convolutional layer, and the layers above will learn to combine them into more complex patterns.

The produced feature map has an important characteristic; it shares the same weights and parameters for this specific filter across the entire input. Sharing weights in this way significantly reduces the number of weights the network has to learn, making it easier to learn very deep architectures and additionally allowing recognizing the patterns in an image to be position independent [25].

2.3.2 Fully Convolutional Networks

Fully Convolutional Networks (FCNs) are CNNs in which the dense layers at the top of the network are replaced by convolutional, fully connected layers. It was first introduced by Long et al. in [26]. The importance of this is that a convolutional layer will easily process an image of any size, whereas a dense layer expects an input of a specific size. And since FCNs contain only convolutional layers, they can be trained on inputs of any size [24].

2.3.3 Region-based Convolutional Neural Networks

In [27], Girshick et al. proposed a technique to use selective search in order to extract a certain number of category-independent regions in an image; an approach they called region proposals. Each region proposal is then warped into a CNN that extracts a fixed-size feature vector. Finally, the feature vector is fed into a Support Vector Machine ⁶ (SVM) to classify the category of the proposed region.

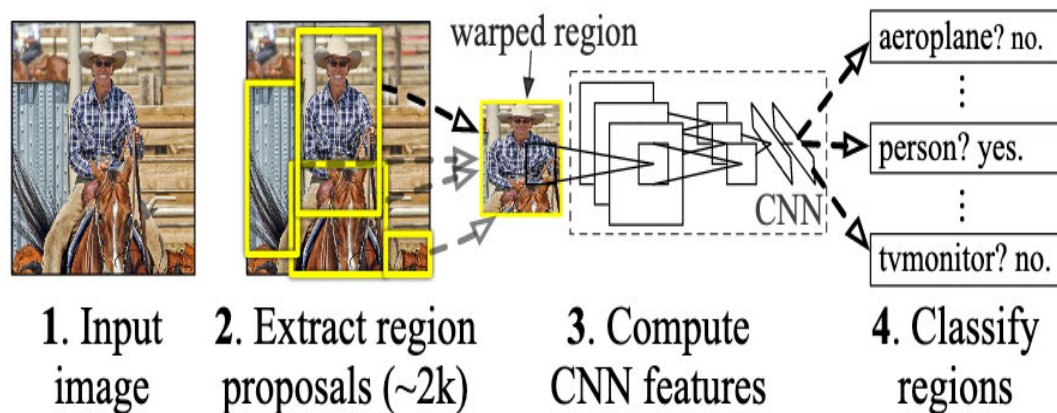


Figure 2.5: R-CNN: Regions with CNN features [27].

⁶A SVM is a versatile ML model that is capable of performing linear or non-linear classification, regression, and outlier detection [24].

2.3.4 Fast-er R-CNNs

Ross Girshick, one of the authors of the R-CNN paper, introduced an improved, faster object detection algorithm to solve some of the drawbacks of the R-CNN in another paper in 2015; the Fast R-CNN. He claimed that R-CNN is slow due to the fact that it generates about 2000 region proposals for each image to feed it to the CNN. Fast R-CNN on the other hand, feeds the input image as a whole to the network in order to extract a convolutional feature map out of it [28].

Alongside the input image, a set of object proposals is also fed into the network and for each of the proposals a region of interest layer then extracts a fixed-length feature vector from the feature map which in turn gets fed into a FCN that finally produces two output layers which indicate the classes of the proposed objects as well as the offset values for the objects' bounding boxes.

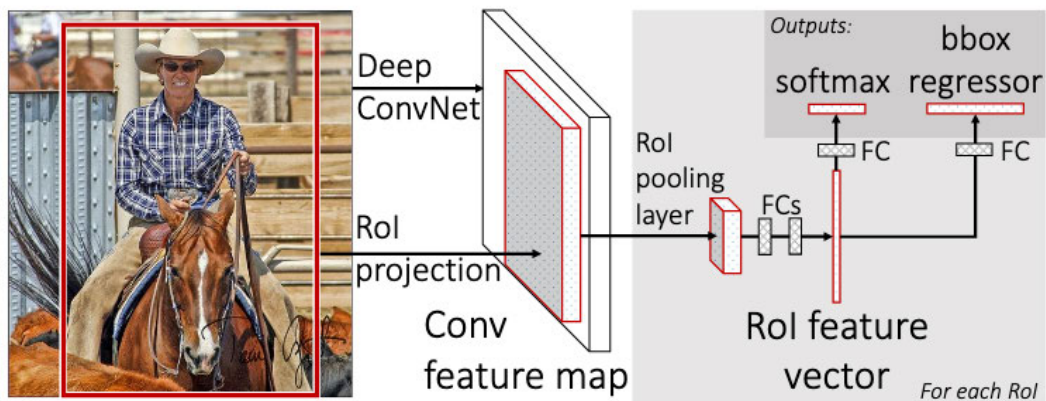


Figure 2.6: Architecture of a Fast R-CNN [28].

In 2016, Ren et al.[29] came up with yet another improvement to make the Fast R-CNN even faster; they literally called it Faster R-CNN. As it has been inferred by Girshick[28], region proposal is an expensive computational operation, and for this reason, Faster R-CNN introduced a network called the Region Proposal Network⁷ (RPN) that shares full-image convolutional features with the detection network, hence

⁷A RPN is a FCN that simultaneously predicts object bounds and objectness scores at each position.

enabling nearly cost-free region proposals. It is trained to generate excellent region proposals. The Faster R-CNN architecture can be seen in Figure 2.7.

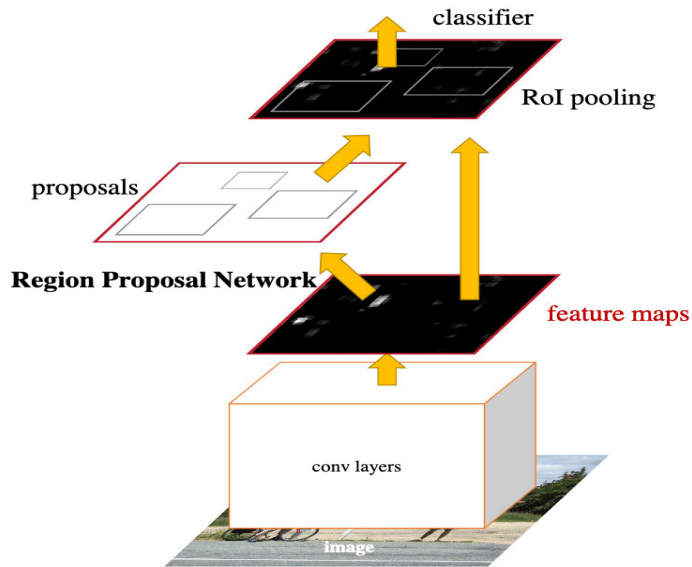


Figure 2.7: Architecture of a Faster R-CNN with the RPN [29].

This RPN is then merged with a Fast R-CNN to form a single network by sharing their convolutional features. The RPN then directs the unified network on where to look. Figure 2.8 shows the significant improvements that have been done over the years to the R-CNNs.

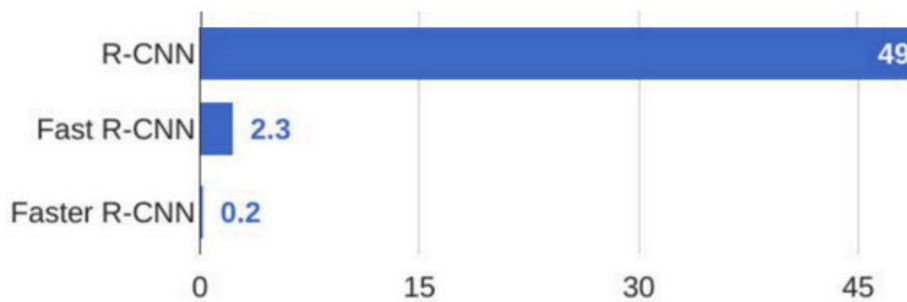


Figure 2.8: Comparison in hours between ordinary, Fast, and Faster R-CNNs trained on the same set of data[30].

2.3.5 Single Shot Detectors

A new, improved object detection methodology using a single deep neural network was introduced by Liu et al. [31] which they called Single Shot Detector (SSD).

As previously discussed, Faster R-CNN uses a RPN to create region proposals and create object bounding boxes and while it has proved to be excelling in terms of accuracy, it is still far below what real-time object detection needs. SSDs on the contrary, eliminate the need of the RPN and use only a single network and only need a single shot to detect multiple objects within the image, hence the name.

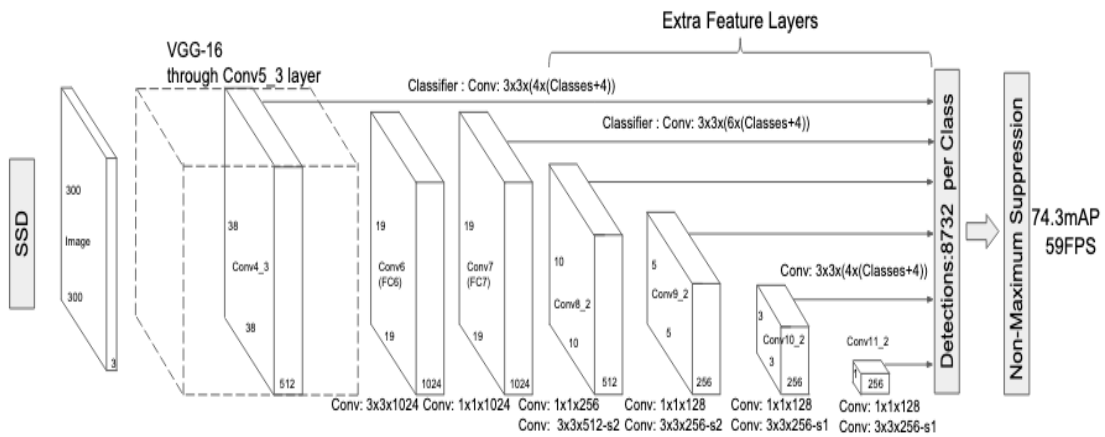


Figure 2.9: Architecture of a SSD object detection model [31].

At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. All computation is encapsulated in a single network which makes SSDs much faster than other approaches.

2.3.6 You Only Look Once

A yet further-improved approach for object detection was introduced by Redmon et al.[32]; an approach they called YOLO. The YOLO network has four -official⁸-incremental versions that will shortly be discussed. YOLO has achieved a huge success in the field of real-time object detection. An overview of the model can be seen in Figure 2.10.

It is considered as one of the fastest, most effective object detection algorithms. A base YOLO model processes images in real-time at 45 Frames per Second (FPS) in addition to achieving very high accuracy prediction rates. The idea is to reframe the object detection as a single regression problem and produce bounding boxes and class probabilities out of the image pixels.

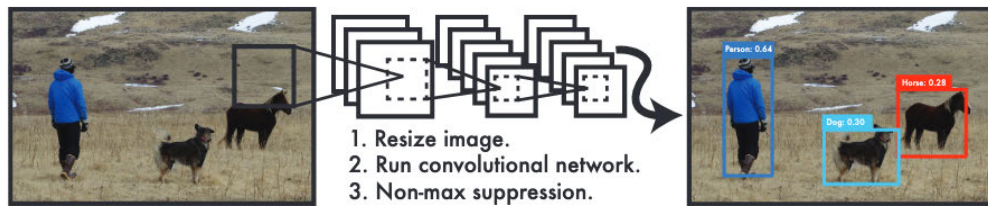


Figure 2.10: The YOLO object detection system at a glance [32].

The concept of a YOLO network is fairly simple; A single CNN simultaneously predicts bounding boxes of objects and class probabilities for such boxes. As shown in Figure 2.11, the architecture of the network is inspired by the GoogLeNet⁹ model and consists of 24 convolutional layers followed by 2 fully connected layers. However, YOLO uses 1×1 reduction layers followed by 3×3 convolutional layers instead of the inception modules used by GoogLeNet. The final output of the network is the $7 \times 7 \times 30$ tensor of predictions.

⁸There is a fifth, unpublished version of YOLO by Glenn Jocher [33].

⁹GoogLeNet is a convolutional neural network that is 22 layers deep used for image classification and object detection. It is able to classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

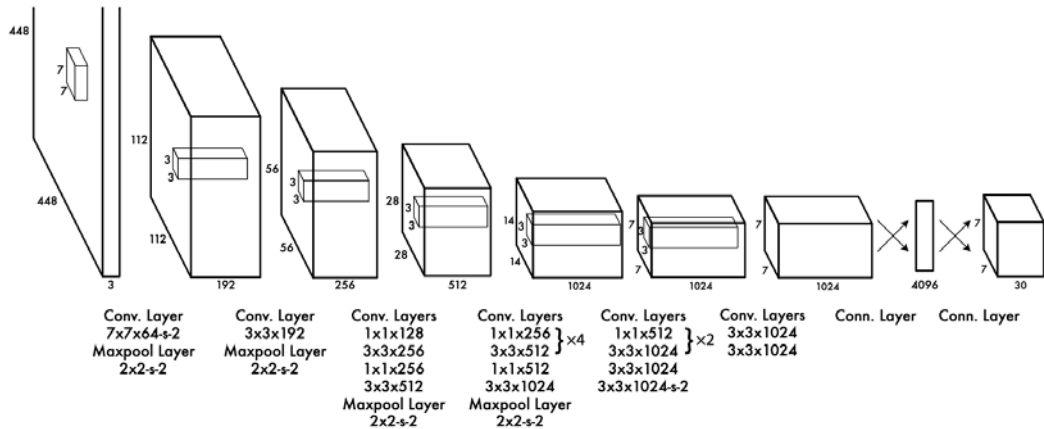


Figure 2.11: Architecture of a YOLO object detection model [32].

The first version of YOLO suffered from several shortcomings relative to state-of-the-art object detection systems. Analysis of YOLO has proved to be erroneous in comparison to Fast R-CNN. Additionally, it has relatively low recall compared to region proposal-based approaches [34]. Therefore, the authors introduced a new version which further improved the system; YOLOv2. They focused mainly on improving recall and localization while at the same time maintaining classification accuracy and speed. They also introduced a real-time framework they called YOLO9000 for detecting more than 9000 object categories by jointly optimizing detection and classification.

The authors of the first two versions did not stop there. In 2018, they were able to introduce a third version with even more updates and improvements; YOLOv3. They trained the network on more data that it became a little bigger and more accurate while maintaining the network speed. As it can be seen in Figure 2.12, YOLOv3 is as accurate as the SSD network but three times faster [35].

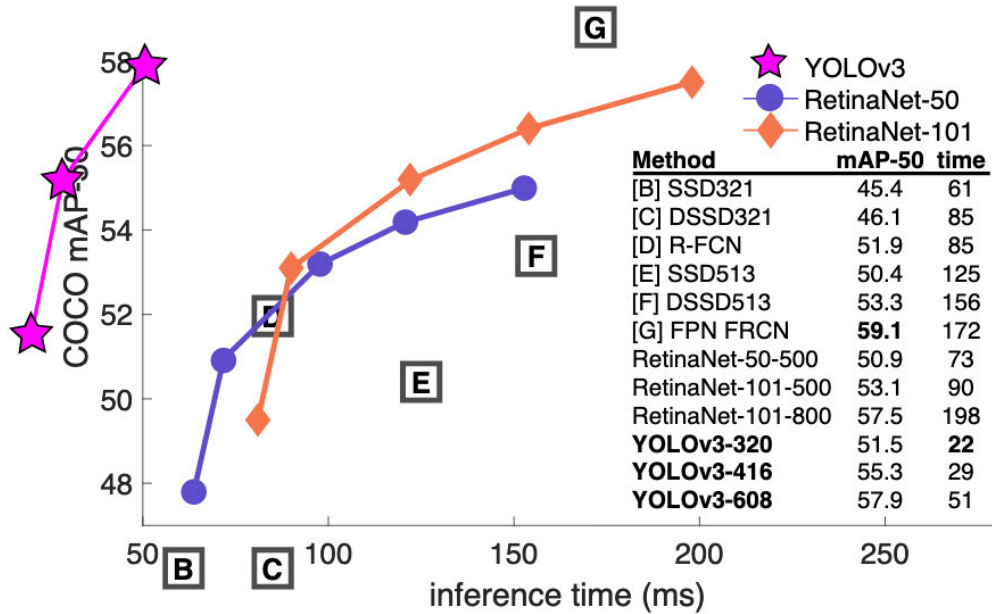


Figure 2.12: Speed/Accuracy trade-off of different state-of-the-art object detection systems [36].

Even though YOLOv3 has had significant success, improvements done to the YOLO model did not stop at the third version. In [37], Bochkovski et al. introduced a further improved fourth version of the network; YOLOv4. They have added numerous features in this iteration that made one of the fastest object detection models even faster. Figure 2.13 shows that YOLOv4 runs twice as fast as EfficientDet¹⁰ with comparable performance. It also has a 10% Average Precision (AP) and 12% FPS improvement over YOLOv3.

Additionally, the new features allowed them to create a model which operates in real-time on a conventional GPU, and for which training requires only one conventional GPU. This was a major problem as the most accurate modern neural networks do not operate in real time and require large number of GPUs for training with a large mini-batch-size.

¹⁰EfficientDet is a recent family of object detectors that achieved high accuracy and reduced the number of flops [38].

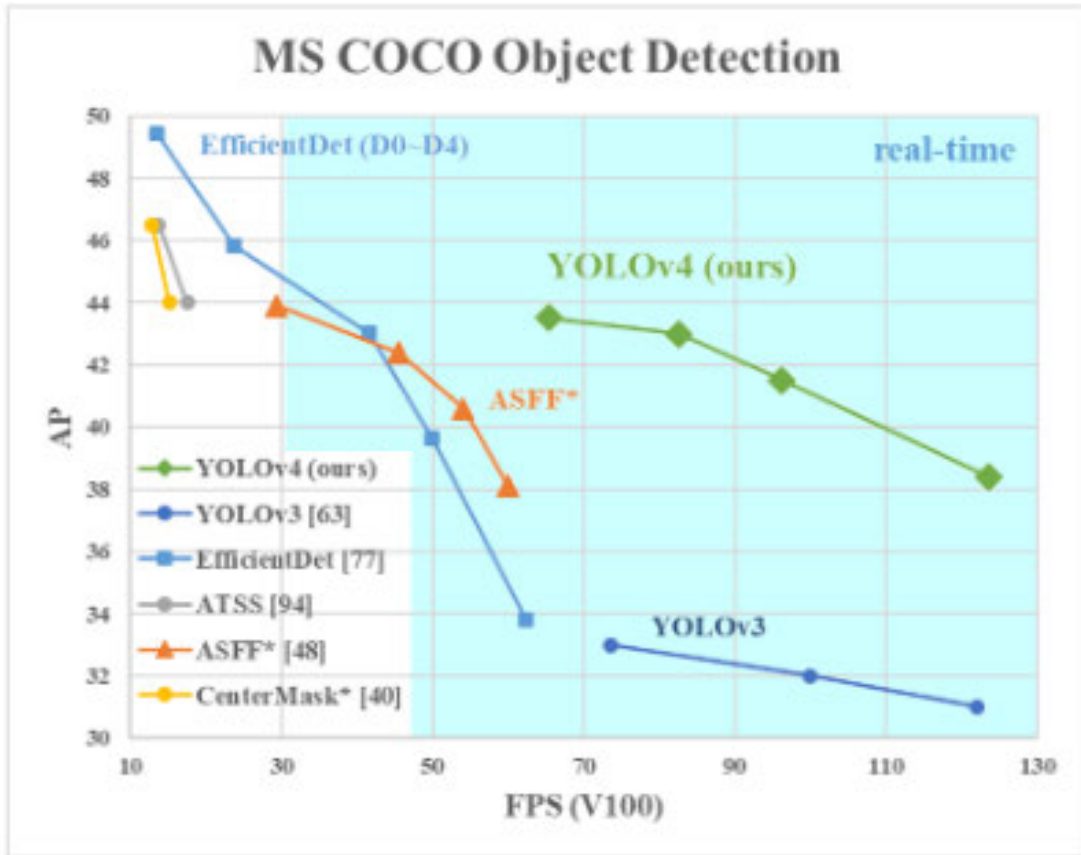


Figure 2.13: Comparison between YOLOv4 and other state-of-the-art object detectors [37].

A few weeks after [37] has been published, an unofficial fifth version of YOLO was released by Glenn Jocher [39]. He introduced a PyTorch version of YOLOv5 with exceptional improvements but has not released an official paper yet. As shown in Figure 2.14, YOLOv5 outperforms all the previous versions in terms of AP; it is nearly as precise as EfficientDet but faster [33].

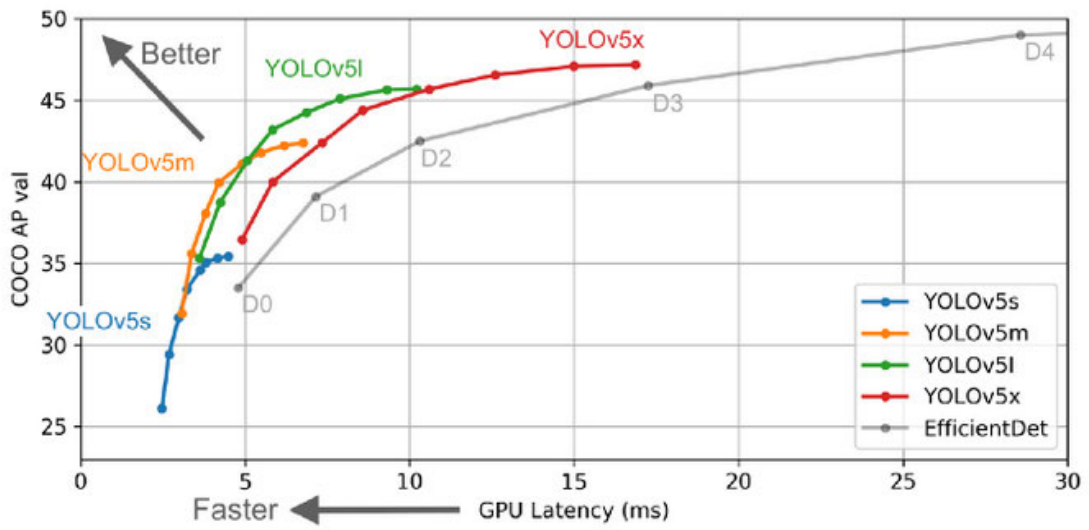


Figure 2.14: Comparison between YOLOv5 and EfficientDet [39].

Now, state-of-the-art methodologies in the field of computer vision and the algorithms that can be used within the scope of this project have been discussed. In the next chapters, the requirements of the project work are going to be analysed in order to come up with an evaluation criteria that is applied at the end of this project.

3 Requirements Analysis

Defining the project's technical requirements is one of the most important steps in project planning in order to deliver a functional product. In this chapter, the specific technical requirements of this project and the important aspects of delivering a functional and reliable model will be defined. Based on the methodologies discussed in the previous section, a clear perspective is now established on what can be used and after defining the project requirements, a decision of which methodologies are used will be made in order to meet the requirements and achieve the highest possible outcome.

3.1 Overview

This thesis aims to develop a machine learning model that identifies garbage in public places from a video stream provided by a drone camera in real-time. The trash detection outcome has to be of high accuracy to make sure that the model does not mistake people or pets for garbage. Additionally, the model needs to be able to identify garbage from an altitude of approximately 5 meters.

Figure 3.1 illustrates the life-cycle of nearly any machine learning model. First, the project requirements and the outcome need to be clearly defined. Next, enough data must be collected to train the model on the specified task. This data has to be prepared in a way the model can understand and make sense of before any training takes place. Additionally, the data has to be divided into three subsets; a training set, a validation set and a testing set.

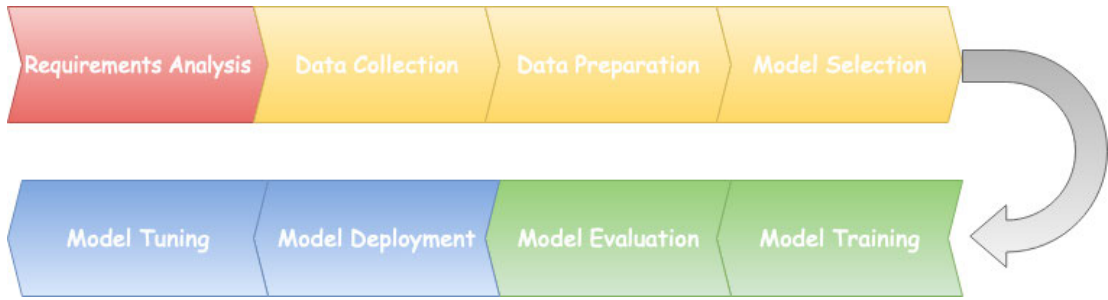


Figure 3.1: The life-cycle of a machine learning project [40].

Splitting the data is very important for evaluating the model’s performance. The validation set is required for adjusting the model’s hyper-parameters during training. However, during testing, the test subset comes in handy as the model has to be tested on data that was never seen before, otherwise, the evaluation becomes compromised and inaccurate. The whole data preparation process can be seen in Figure 3.2.

Data Preparation Process

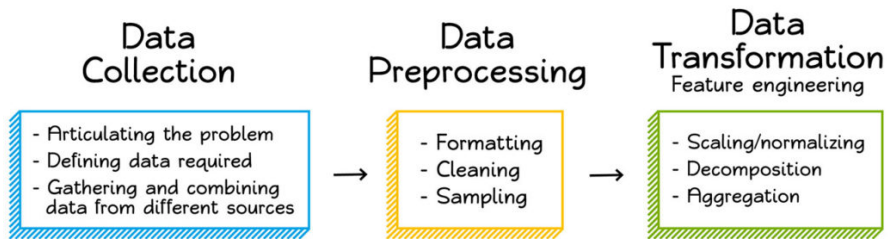


Figure 3.2: Data preparation process [41].

An appropriate model is then chosen and the training process takes place. The model is automatically tweaked and enhanced by the used neural network throughout the learning phase and is then validated using the validation data. Finally, the model

is deployed and, later on, enhanced again and tuned to perform better and produce a more superior outcome.

3.2 Functional Requirements

First and foremost, the work of this thesis must produce a functional model that is able to detect garbage in public places and can later be improved to provide better outcome. Since this model is going to be run on a drone with low computing power, the model needs to be lightweight and has to have the ability to run on a standard laptop.

Moreover, the detection will take place in real-time, and thus, the used algorithm has to prioritize the detection speed over any other parameter. The resulting model has to run with at least 10 frames per second to satisfy the real-time detection constraint. This constraint will also play a huge part in choosing the network that will be used for implementing the model; this will be discussed in detail in the next chapter.

Even though accuracy is not considered the most important parameter within the scope of this project, it still plays a very important role that shall not be neglected. It has to be ensured that the model satisfies a certain threshold when it comes to detection accuracy. The model shall not classify people, pets nor objects that usually exist in a public places (plants, trees, etc.) as garbage. A summary of the defined functional requirements can be seen in Table 3.1.

Priority	Requirement
1	Model is functional
2	Detection has to take place in real-time
3	Model is able to perform detection at a speed of 10 FPS on a standard laptop (without GPU)
4	Model can only detect garbage and not other objects
5	Garbage detection is accurate
6	Model is lightweight
7	Model is able to detect garbage from altitude of approximately 5m

Table 3.1: Summary of the project's functional requirements.

3.3 Non-functional Requirements

Machine learning algorithms learn from data. It is crucial to feed them the right data for the problem at hand. It is also very important to make sure the data at hand contains the right features and is pre-processed in a way the model can understand and make sense of. The more relative the data that gets fed to the model, the better the outcome becomes.

Furthermore, computer vision tasks tend to require a relatively large amount of data to be able to perform accurate detection or classification. As it can be seen in Table 3.2, some architectures are trained on datasets that contain over 1.2 million images and can classify up to 1000 categories of objects. Hence, a large dataset containing images of garbage in public places has to be present in order to produce the required model with high precision rate. The images have to be of size 416×416 (this is discussed thoroughly in the development chapter).

Name	Dataset size used for training (# images)
Le-Net5	70 K
AlexNet	1.2 M
VGG16	1.2 M
VGG19	1.2 M
GoogLeNet	1.2 M
ResNet-50	1.2 M
DenseNet	1.2 M

Table 3.2: Popular CNN-based architectures used in computer vision applications and the sizes of the datasets used for their training [42].

Another important requirement is the availability of a powerful Graphical Processing Unit (GPU) during the training process. Usually, training a computer vision algorithm can be very computing-heavy and may take up to days if not weeks depending on the size of the dataset hence the need for a powerful GPU during training in order to speed up the process.

On the development side, the used ML frameworks and computer vision libraries are chosen according to the aforementioned requirements. Preferably, one tool is used for as both ML framework and a computer vision library. Additionally, they must also have good community support and reasonable implementation effort. Within the scope of this thesis, open-source projects are preferred over paid tools. Table 3.3 summarizes the project's non-functional requirements.

#	Requirement
1	Large dataset available
2	Images are of size 416×416
3	Powerful GPU available during training process
4	Programming language must have large support for 3rd party libraries and has to be suitable for machine learning and computer vision applications
5	One tool for ML & computer vision
6	Used tool is open-source
7	Used tool has good community support and reasonable implementation effort

Table 3.3: Summary of the project's non-functional requirements.

In the next chapter, the used technologies and tools are thoroughly discussed and compared against the requirements defined in this chapter.

4 Conception

Choosing the technologies used within the project and deciding between available approaches is a major step in the project life-cycle. In this chapter, the methodologies used to achieve the project requirements defined in the previous chapter are discussed as well as the implementation alternatives for the system and which approach fits the requirements the best.

4.1 Network Architecture

In the previous chapter, the requirements needed for this project to come to life were defined. It was made very clear that after delivering a functional model, the most important requirement of all is the speed of the model. Given that the detection will take place in real-time, the chosen network architecture shall be the fastest one available. However, the detection accuracy shall also not be compromised as it plays a decent role in the model development.

Based on the work of Redmon et al.[9] and as discussed in section 2.3.6, YOLOv3 proved to be as accurate as many of state-of-the-art object detection networks but much faster. However, this is not the only reason that this approach is chosen for fulfilling the work of this thesis. As it can be seen in Table 4.1, YOLOv3 is superior in other aspects as well.

Network Architecture	Tiny YOLO	YOLOv2	YOLOv3	YOLOv4	YOLOv5	SSD	Faster R-CNN
Point of Comparison							
Detection speed	Superior	Neutral	Superior	Superior	Superior	Neutral	Inferior
Detection accuracy	Inferior	Inferior	Neutral	Superior	Superior	Neutral	Superior
Lightweight network	Superior	Inferior	Superior	Neutral	Neutral	Neutral	Inferior
Community support	Superior	Neutral	Superior	Inferior	Inferior	Superior	Inferior
Implementation effort	Superior	Neutral	Superior	Inferior	Inferior	Neutral	Superior

Table 4.1: Comparison of available state-of-the-art object detection architectures.

Not only can YOLOv3 perform the detection faster than other approaches without compromising the accuracy, it is also widely used for many object detection systems worldwide. There is a plethora of resources that explain the concept and how to adapt it to fit one's own needs. Additionally, plenty of GitHub¹ repositories containing different implementations of the network using different ML frameworks and computer vision libraries can also be found.

The approach is simple to implement and the authors provide a way to run a trivial object detection model in a matter of minutes on their website as a proof of concept [43]. The trivial model uses pre-trained YOLO weights that are trained on the COCO dataset². However, further steps can be taken in order to develop a custom object detection model for more challenging tasks.

4.2 Hardware

4.2.1 Training Hardware

As it has been discussed in section 3.2, a powerful GPU has to be used for training a computer vision model as processing images is a computation-heavy process and cannot be performed on a normal device without a GPU if a large dataset is being used. Hence, a powerful GPU available on the premise of the Hamburg University of Applied Science was used to perform model training. It was available via accessing a Virtual Machine (VM) provided by the university.

¹GitHub is a platform that provides hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.

²COCO is a large-scale object detection, segmentation, and captioning dataset with around 330K images containing 80 different object classes [44].

4.2.2 Detection Hardware

For the purpose of detection, it has been defined in this project's functional requirements (see Table 3.1) that the detection has to take place on a standard laptop with low computing power. The output model has to run at the speed of at least 10 FPS on a machine without a GPU.

4.3 Software Technologies

In this section, the software technologies used to make this project come to life are elaborated. A comparison of the most used ML frameworks and computer vision libraries is done in addition to choosing the project's programming language.

4.3.1 Programming Language

There is a huge variety of programming languages nowadays and almost any of them can be used for ML applications. However, writing every algorithm from scratch is a time consuming and error-prone process. This is why the best-suited programming language is the one that comes with a lot of libraries and has advanced support for AI applications. According to a report by GitHub[45], the top 3 programming languages for ML in 2018 were Python, C++ and JavaScript respectively.

Python is one of the most popular programming languages of recent times. In addition to being the first language used for ML, it is also the third most used language in projects on GitHub. Moreover, Python was one of the first programming languages to get the support for ML via a plethora of libraries and tools that make AI applications development extremely easier for engineers. In addition to being easy to use, the neural network and computer vision libraries used in this project -more on that in the next section- also come with Python support which is why it has proved to be the most fit programming language for this project's requirements.

4.3.2 Machine Learning Framework

Machine Learning is one of the fastest emerging technologies today and this is the reason behind the large number of ML frameworks that are available. A Machine Learning Framework is an interface, library or tool which allows developers to build machine learning models easily, without getting into the depth of the underlying architecture and the hassle of implementing everything from scratch. Engineers tend to look for the most suitable framework for their various kinds of ML projects. A few of those frameworks are thoroughly defined in the following sections.

TensorFlow

TensorFlow is an end-to-end open source platform for machine learning built by Google. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML as well as giving developers the ability to easily build and deploy ML powered applications [46].

TensorFlow has been becoming more and more popular over the past couple of years due to its power and ease of use and is considered one of the most used ML and DL libraries of this time. It lets developers bring the power of DL to computer vision applications and has some great tools and libraries to perform image processing and classification.

Keras

Developed by a Google engineer, François Chollet, Keras is an open-source neural network library designed to be a high-level Deep Learning wrapper around various libraries like TensorFlow or Theano³ in the backend. It strives to provide a simple Application Programming Interface (API) for working with neural networks and computer vision applications [12].

³Theano is a Python library that allows for defining, optimizing, and evaluating mathematical expressions involving multi-dimensional arrays efficiently.

Keras makes it simple for ML beginners to design and develop a neural network and DL applications. In addition to its simplicity and ease of use, Keras is also more mature in terms of community support, available tutorials and the ability to use TensorFlow in the backend.

PyTorch

PyTorch is an open source deep learning framework built to be flexible and modular for research, with the stability and support needed for production deployment. It provides a Python package for high-level features like tensor computation with strong GPU acceleration and TorchScript for an easy transition between eager mode and graph mode [47].

PyTorch is developed by Facebook and is also a huge competitor to TensorFlow in terms of popularity. It is mainly used to train deep learning models quickly and effectively and this is why it is especially popular in the research community.

Conclusion

It is a tough decision to favor any of the previous frameworks over the other since they are all very powerful in terms of fulfilling the project's requirements. However, using Keras (in combination with TensorFlow in the backend) is the final decision due to the fact that they are one of the most required skills in today's job market for the field of AI and computer vision. A full comparison between the aforementioned ML frameworks can be seen in Table 4.2.

Point of Comparison	Framework			
		TensorFlow	Keras	PyTorch
Open source		✓	✓	✓
Python integration		✓	✓	✓
Computer vision functionality		✓	✓	✓
Easy to use		✓	✓	✓
Community support		✓	✓	✓
Low implementation effort		✓	✓	✓

Table 4.2: Comparison of the most popular ML and DL frameworks.

4.3.3 Computer Vision Library

Over the past decade, tools for computer vision have gained noticeable progress in terms of new libraries and their power. Moreover, improvements in hardware like GPUs as well as the ML tools and frameworks have made computer vision much more powerful in the current time. In this section, some of those available tools are discussed in order to find the best one for the project requirements.

Keras API for Computer Vision

As it was mentioned in section 4.3.2, Keras is mainly a DL library that provides extensive features for developing neural networks and DL application. It is designed to be simple and easy to use even for beginners in this field. Additionally, Keras provides a lot of features for computer vision applications and is widely used for object detection networks due to its speed, power and low implementation effort.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Its features allow for processing images and video frames, detection of certain patterns, identification of required elements present in the photo or video and classification of objects [13].

Although OpenCV is a powerful and widely-used library, it would be impractical to use within the scope of this project as there would be a need to use different tool as a machine learning framework. Additionally, the implementation effort of training a YOLO model in OpenCV is higher than doing it using Keras.

Matlab

Matlab is a great tool for creating image processing applications, data analysis and visualization and it is widely used in research. It has its own user-friendly Integrated Development Environment (IDE) which allows for quick prototyping, error tracing, debugging and a lot of other features. Matlab is heavily used in research topics and prototyping and its code is quite concise making it easier to read and debug [48].

Despite being a powerful library for computer vision, Matlab is a paid tool and can also get relatively slow during execution time. As compared to OpenCV and Keras computer vision API, Matlab fails to meet the requirements for the scope of this work.

ImageJ

ImageJ is an open source image processing program designed for scientific multidimensional images. It is highly extensible, with thousands of plugins and scripts for performing a wide variety of tasks, and a large user community. User-written plugins make it possible to solve almost any image processing or analysis problem, from three-dimensional live-cell imaging, radiological image processing, multiple imaging system data comparisons to automated hematology systems [49].

ImageJ was designed with an open architecture that provides extensibility via Java plugins. Custom acquisition, analysis and processing plugins can be developed using ImageJ's built in editor and Java compiler. Despite its strength, ImageJ is mainly used in scientific research and requires high implementation effort for the scope of the problem at hand. It is also poor in terms of community support and Python 3 integration.

Conclusion

When comparing the aforementioned computer vision libraries it can be seen that only of two of them stand out and are able to fulfill this project's requirements; Keras API for computer vision and OpenCV. It would not be fair to favor one of them over the other as the two are extremely powerful and heavily used across computer vision applications. However, Keras meets one extra requirement than OpenCV which is using one tool as a ML framework and a computer vision library. This is why Keras was chosen as a better fit for this project. A complete comparison of the aforementioned libraries can be seen in Table 4.3.

Point of Comparison	Library	Keras	OpenCV	Matlab	ImageJ
Open source		✓	✓	✗	✗
Python integration		✓	✓	✗	✓
One tool for ML & computer vision		✓	✗	✗	✗
Easy to use		✓	✓	✓	✗
Community support		✓	✓	✗	✗
Low implementation effort		✓	✓	✓	✗

Table 4.3: Comparison of the most popular computer vision libraries.

5 Development

In this chapter, the technical details of this project's implementation are discussed. First, the used data and how it is collected and prepared for training is elaborated. Afterwards, the implementation details of the model and the training process are discussed.

5.1 Dataset

The significance of data in developing any machine learning algorithm has already been established. Machine learning algorithms learn from data, thus, they should be fed the relevant data for the task at hand in order to perform well when deployed in real life situations. If the model is fed irrelevant data, it would not be able to extract features that make sense out of it, which in turn will produce a model that behaves in other ways than expected.

It has also been elaborated how object detection applications usually tend to require large amounts of data to produce a decent model in terms of accuracy. A functional model can be produced with a very small set of data, however, it cannot be expected to perform with high accuracy on objects that it had not seen very often.

Since the model developed within the scope of this work shall not compromise detection accuracy, data is collected from two different datasets and merged together to form one larger dataset to ensure data sufficiency for the model.

5.1.1 Data Collection

Finding the relevant data for a specific machine learning problem is usually the hardest part of the project especially when the problem at hand is infrequent as the one discussed in this thesis. This is why data had to be collected from two different datasets.

The first source of data is a dataset called trashnet collected by Thung et al. for their machine learning class project at Stanford University [50]. The dataset consists of roughly 2,500 images of size 512×384 containing nearly 10 different object categories. And although their dataset covers a whole lot of different garbage categories, the images have been taken from a low altitude and in a closed place as it can be seen in Figure 5.1. This is the reason why extra data had to be collected from a different source.

Since the required model needs to perform well from higher altitudes and in public places, additional images of garbage in those circumstances are collected from another source; the previously mentioned Let's Do It Foundation trash dataset [51]. For their project, they had collected a huge dataset of roughly 28,000 images of trash. However, most of those images are irrelevant to this project as they are sometimes in seas and oceans and other times full of trash as in garbage disposal places.

The dataset is filtered only to the images relevant for the project's purposes; images from a high altitude and that contain garbage as part of them. After dataset filtration, a small set of roughly 1,000 images were collected and merged to the first source of data to achieve a dataset of approximately 3,500 images. An example of the selected images is shown in Figure 5.2.

Finally, all the images are resized to the size of 416×416 as the model expects the data to be of that size. For that purpose, a python library, `python-resize-image`[52], is used to get the desired image sizes.

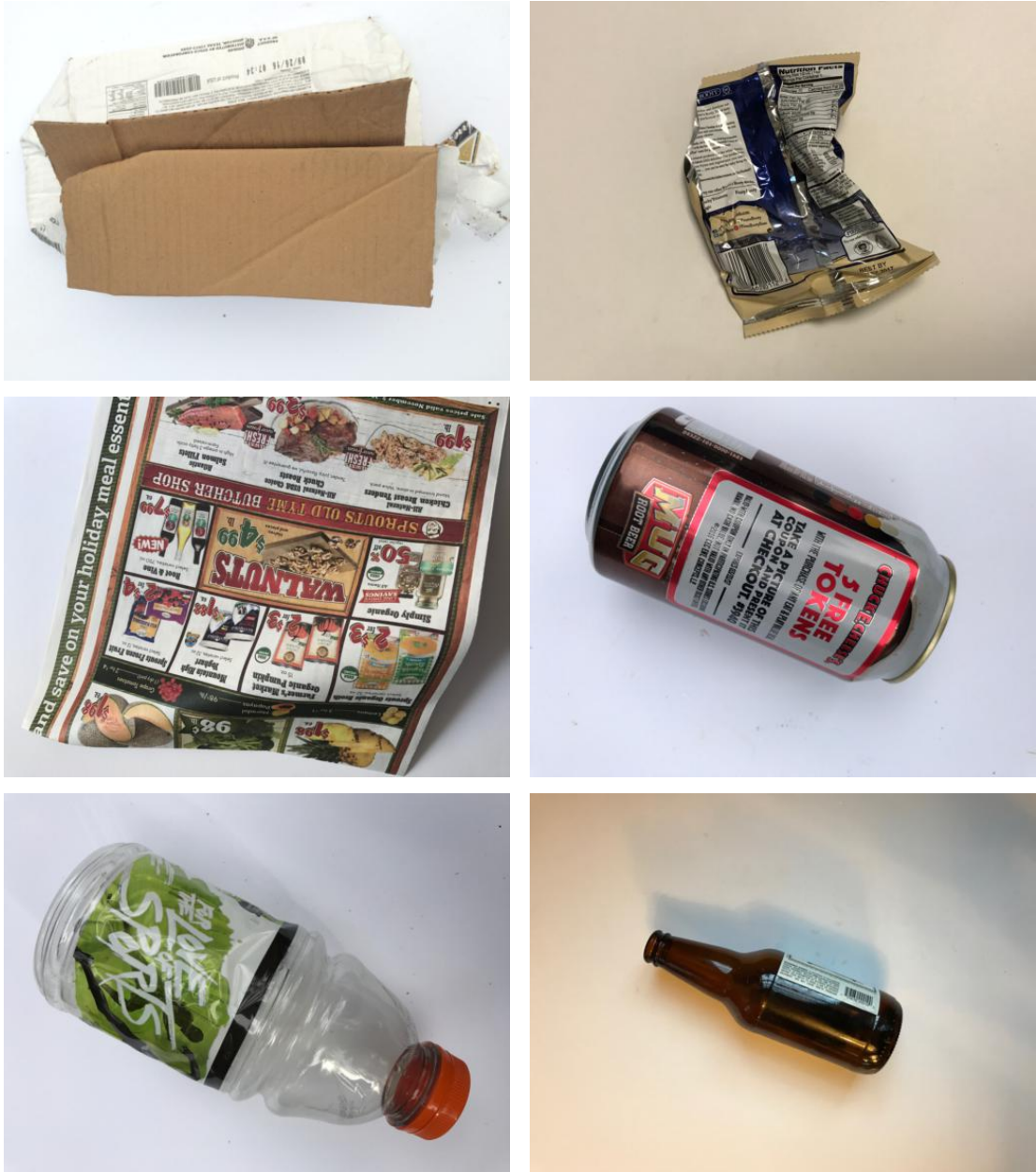


Figure 5.1: Samples of images from the trashnet dataset [50].



Figure 5.2: Samples of collected images from the Let's Do It Foundation dataset [51].

5.1.2 Data Pre-processing

In addition to resizing the images to optimize training time, another measures are taken in order to obtain a larger dataset and prepare the data for training. Data augmentation is performed on the dataset in order to get more variation in the existing images. Additionally, all the images in the dataset have to be annotated to indicate the garbage objects within the image.

Data Augmentation

Data augmentation artificially increases the size of the dataset by generating many variants of each instance in it. Those generated variants should be as realistic as possible. Ideally, given an image from the augmented set, a human should not be able to tell whether or not it was augmented. Additionally, the modifications done on the original dataset have to be learnable; the images can be shifted, rotated, resized or contrast-manipulated. This forces the model to be more tolerant to variations in the position, orientation, light conditions and size of the objects in the images. An example of how augmentation can change the images can be seen in Figure 5.3.

By combining these transformations, the size of the dataset can be greatly increased. This in turn drastically reduces overfitting; that is when a model learns the details in the training data to the extent that it negatively impacts the performance of the model on new data. This makes the model achieve outstanding outcome on the training data and poor results on never before seen data [24].

It was mentioned that the final dataset used for this project contains approximately 3,500 images. Data augmentation is performed to introduce rotations, shifts, shearing and contrast to the original images in the dataset. For that purpose, a python library called `imgaug`[53] is used. After introducing data augmentation, the dataset became twice the size and ended up containing approximately 6,500 images.



Figure 5.3: An example of augmented images (right) in comparison to original images (left).

Data Annotation

Data annotation is the process of labeling the objects of interest in various types of data like text, images and videos while ensuring the accuracy to make sure it can be recognized by the machines through computer vision. For almost any ML project, labeled datasets are required so that the model can easily and clearly understand the input patterns. Moreover, to train a computer vision model, the data needs to be precisely annotated using the right tools and techniques.

With respect to image annotation, there are different annotation types that can be used to indicate the relevant objects in the image. There are some popular methods such as using 2D bounding boxes, polygon annotation, semantic segmentation, landmark annotation, polylines annotation and 3D point cloud annotation. Those types can be seen in Figure 5.4.

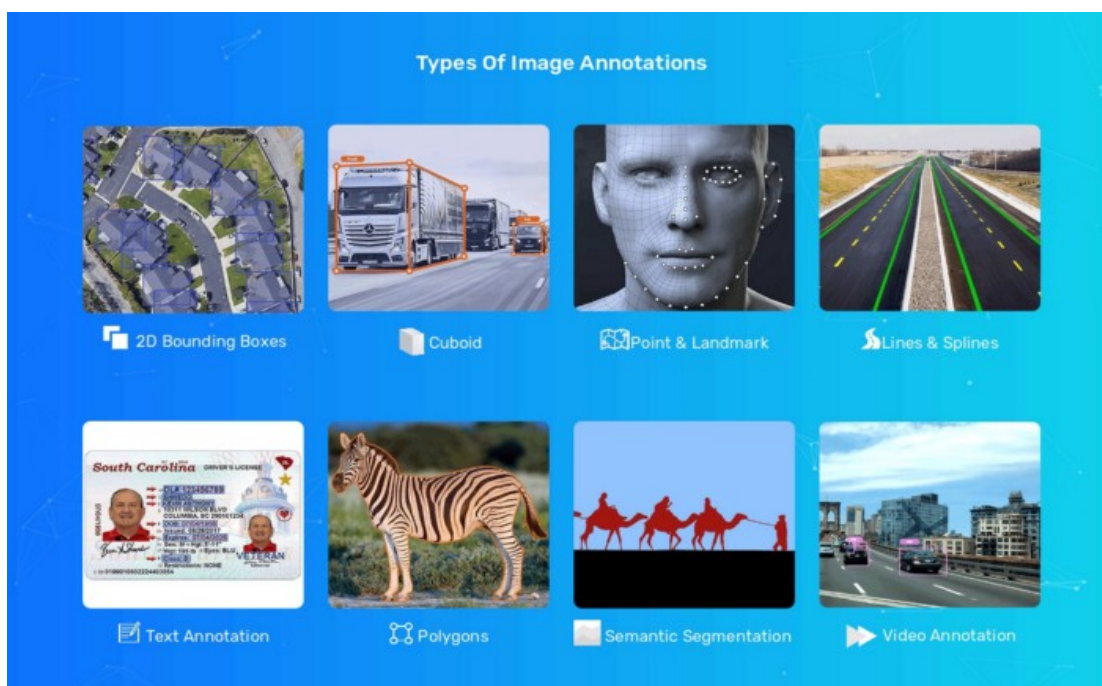


Figure 5.4: Some of the popular methods used for image annotation [54].

Since the images used in this project are not complicated and don't contain multiple objects, using 2D bounding boxes is sufficient to produce a decent model. For the purpose of image annotation, Microsoft's Visual Object Tagging Tool (VoTT)[55] is used. The program has a friendly and easy user interface in addition to being open source.

Splitting the Data

As already discussed in section 3.1, splitting the data is crucial for evaluating the model; the evaluation must be based on data that model has never seen before. Otherwise, if the model is tested on the same training data, the integrity is compromised as it will produce good results because it has already memorized the training data. Additionally, a subset of the dataset has to be set aside for validation; that is the set required for adjusting the model's hyper-parameters during training. In this project, the data is split into 7:1:2 for training, validation and testing respectively.

5.2 Model Implementation

YOLOv3 is one of the most widely used object detection algorithms. Hence, it has one of the largest online communities that provide a lot of useful implementations to the network. Plenty of GitHub repositories provide different implementations to the algorithm using different libraries and frameworks and this is a huge advantage making YOLOv3 superior to many other algorithms. Despite the availability of such repositories and implementations, there is almost always some extra features that need to be added to the existing code depending on the project's requirements.

In this project, a GitHub repository which provides an implementation of YOLOv3 using Keras and TensorFlow is used. The repository[56] is developed by Anton Muehle-
mann and provides an easy step by step tutorial on how to use and customize it to the needs of each project. However, the code had to be adjusted and tweaked in some ways in order to fit the needs of this project.

Upgrading TensorFlow Version

At the time of writing and implementation, the original code was using an early version of TensorFlow library (version 1) which caused a lot of errors due to the use of deprecated methods in addition to being impractical. The code had to be updated to use TensorFlow 2. This was done by eliminating all the deprecated methods and introduced errors of using version 1. Additionally, using TensorFlow 2 is more relevant to today's job market.

Using Pretrained Layers

It is generally not a good idea to train a very large DNN from scratch, instead, it is better to find an existing network that accomplishes a similar task to the one at hand and reuse its lower layers; this is called transfer learning. This will not only speed up the training drastically, but also require significantly less training data. This is why engineers usually tend to reuse already existing networks. However, some measures need to be taken in order to adjust the network to the project's needs.

First, the output layer of the network shall be replaced by a new layer since it is likely not useful for the new task and may even not have the right number of outputs of the new task. Additionally, the upper hidden layers are also of low significance to the new task, since the high-level features of the pretrained network may differ to those of the new task. However, the lower layers are really important and can be reused the help improve the model's accuracy and performance. This can be done by freezing the lower layers; that is preventing their weights from being changed during training [15][24].

The more similar the tasks are, the more lower layers that can be reused. Similarly, the more training data available, the more layers that can be unfrozen. In this case, only the top three layers of the model are unfrozen in the beginning and training is carried out until the results stop improving. Afterwards, all the layers are unfrozen to allow for a last step of tuning the model's hyper-parameters and weights and improve the model's accuracy even more. An example of unfreezing all the layers can be seen in Listing 5.1.

```
1 for i in range(len(model.layers)):
2     model.layers[i].trainable = True
```

Listing 5.1: Unfreezing all the layers of the model.

Optimizing Learning Rate

Finding a good learning rate is important because if the model's learning rate is too high training may diverge, however, if it is too low, the training will take a large amount of time. Hence, a trade off has to be made. In this project, the learning rate is varying throughout the training process. The training first starts with a higher learning rate and monitors the model's validation loss. Once the performance stops improving for three epochs, a callback is called and the learning rate is decreased by 0.1 factor. This tremendously helps in improving the model's performance and the hyper-parameter tuning process. Reducing the learning rate callback can be seen in Listing 5.2.

```
1 reduce_lr = ReduceLRonPlateau(monitor="val_loss", factor=0.1,
    patience=3, verbose=1)
```

Listing 5.2: Reduce learning rate callback.

Model Training

After defining the learning rate and the pretrained layers that will be used, the model training process takes place. First, the model is trained with the frozen bottom layers until it reaches a stale point of no improvement for a few epochs. Then, a callback is used to unfreeze the bottom layers of the model to further fine-tune the hyper-parameters. The model has to be compiled again and another Keras callback is used to finally stop the training when the results have stopped improving for a few epochs. This helps in avoiding overfitting during the training process. Due to the limitation of computation resources and the immensity of the dataset, the training process takes approximately 3 days. An overview of the training process can be seen in Figure 5.5.

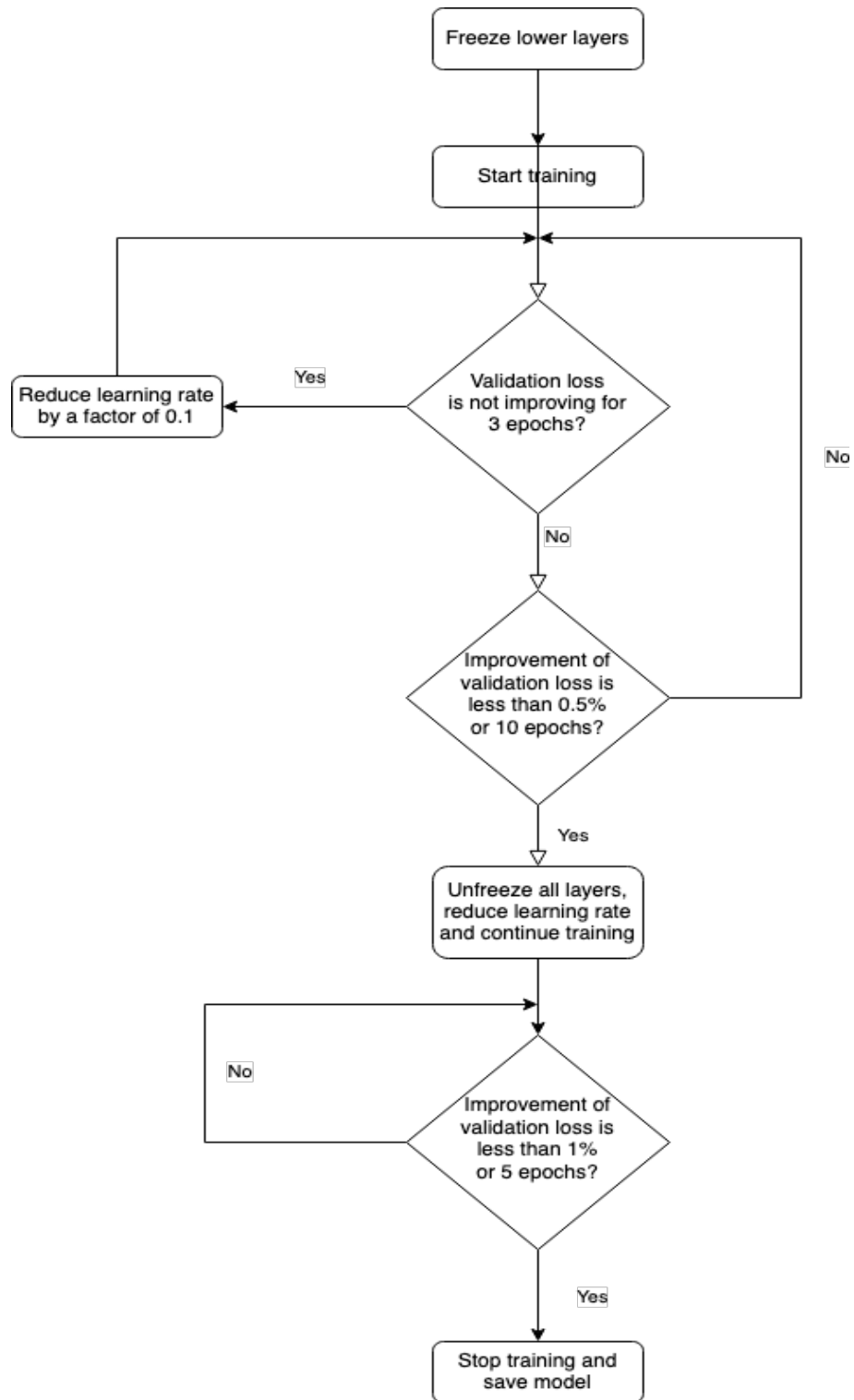


Figure 5.5: Flowchart of the training implementation.

Testing the Model

After the training process is finished, the model is saved in addition to its weights and history; that is the values of the monitored metrics throughout the training process. Additionally, the code also contains a python script that can be run on test images in order to test the model. The script creates output images with bounding boxes around the objects that the model predicts to be trash. The model outcome and evaluation will be discussed thoroughly in the next chapter.

6 Evaluation

In this chapter, the performance metrics used for evaluating the model are discussed. Additionally, different experiments and analyses of the model are done depending on tweaking the model's parameters. The final outcome is also evaluated and discussed for different scenarios of implementation.

6.1 Evaluation Metrics

There are many metrics that can be monitored for evaluating machine learning algorithms, however, there are some specific, very important metrics that are relevant for each different task. Monitoring the training and validation losses are essential to almost any machine learning model. Monitoring the Mean Absolute Error (MAE) is also a good idea for most of the models. Additionally, the Average Precision metric is one of the most used metrics for evaluating computer vision algorithms.

The MAE measures the average magnitude of the errors in a set of predictions. This means that it calculates the average over the test samples of the absolute differences between predictions and actual observations where all individual differences have equal weights. It is usually a good idea to monitor the MAE of the model to get an idea of how its prediction accuracy has improved.

As mentioned above, a very important metric to monitor for computer vision models is the AP, sometimes also called mAP; the Mean Average Precision. This metric is actually a combination of two things, precision, recall. The precision measures how accurate the predictions are, i.e. the percentage of correct predictions. The recall

measures the true positive rate of the model, i.e. how good the model finds all the positive values.

There is almost always a trade-off between recall and precision; the higher the recall, the lower the precision. But there are sometimes a few points where the precision goes up and the recall also increases, especially at low recall values. This can be seen in Figure 6.1. So it simply makes sense to use the point with high precision and high recall. The AP computes the maximum precision at different recall values and then calculates the mean of all those values. Now if the model can predict more than two classes, the AP is computed for each of them and then the mean of those APs is computed; that is the mAP [24].

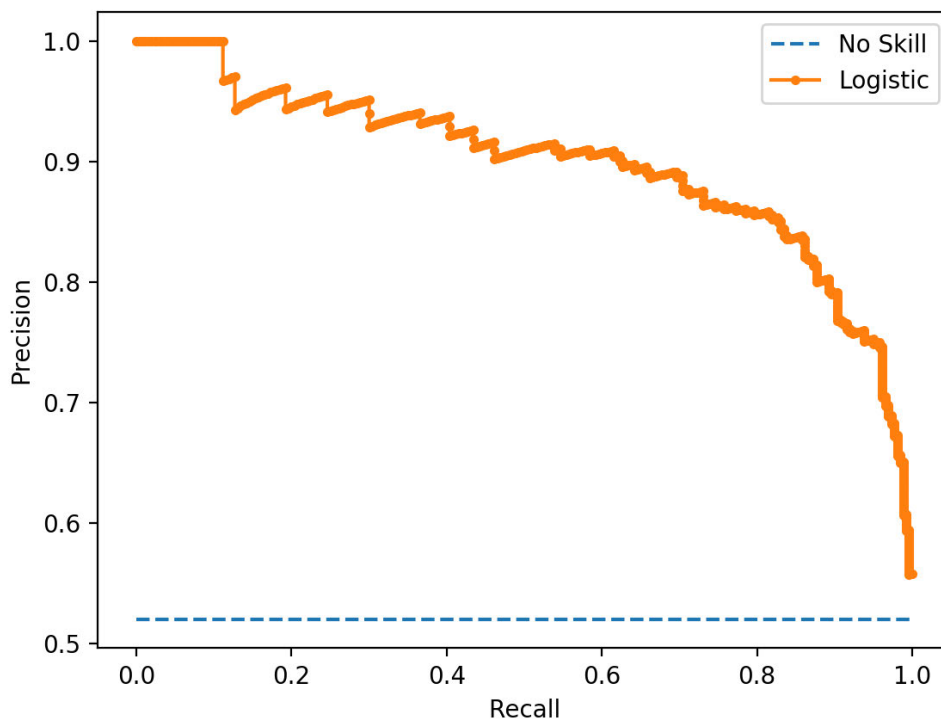


Figure 6.1: Precision vs Recall curve [24].

For object detection systems there is an extra level of complexity; that is if the model predicts the right class but at a totally different location. This shall not be considered a correct prediction and that is where the Intersection over Union (IoU) comes in handy. The IoU measures the overlap between 2 boundaries; that is the overlap between the predicted bounding box and the true bounding box. Usually, an IoU threshold is defined and the mAP is measured for that specific threshold. In some object detection competitions the threshold is 0.5 (50%) and the mAP is noted as mAP@0.5 or mAP@50%. In other competitions, the mAP is computed at different thresholds and the final metric is the mean of all these mAPs [24].

Another important aspect that needs to be evaluated for this project is the speed, in other words, at how many FPS is the model able to run. For this, detection has to run separately on the test dataset and the taken time for detection has to be calculated. Those are the evaluation metrics used for this project's work. In the next section, the experimentation results are discussed and the output is evaluated.

6.2 Experimentation and Results

It is important to try various implementation scenarios and experiment a few times with the model parameters in order to find the best outcome for the problem at hand. It is not the best idea to train the model one time and go along with the produced outcome. In this section, the different scenarios tried during this project work are going to be evaluated.

Using Constant Learning Rate

The first time the model was trained, a fixed learning rate was set while freezing the bottom layers. The learning rate for the top unfrozen layers was not decreased as discussed in section 5.2. However, the learning rate is decreased by factor of 0.1 after unfreezing the bottom layers for the final fine-tuning step. Although this produced a decent model, there was still room for improvement. The MAE was not monitored during this training iteration, only the validation and training losses which can be

seen in Figure 6.2. The model achieved a training and validation loss of approximately 17%.

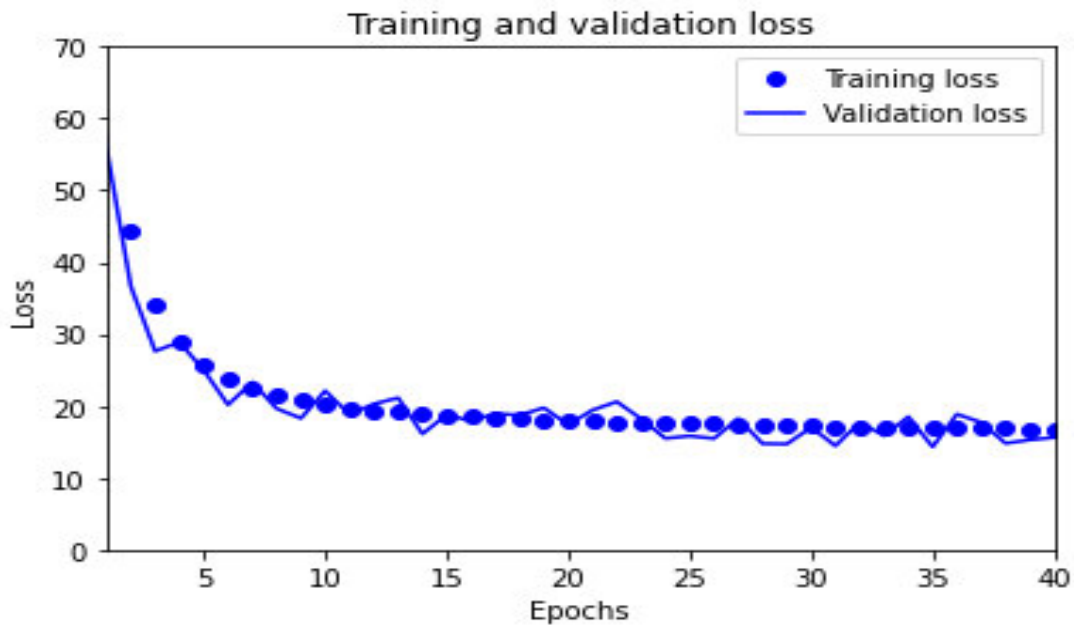


Figure 6.2: Training vs Validation losses during first training iteration.

Optimizing Learning Rate

A second iteration is performed while adjusting the learning rate for the top, unfrozen layer. As the flowchart in Figure 5.5 shows, the learning rate is decreased by a factor of 0.1 if the validation loss stops improving for 3 epochs. This improved the model validation and training loss (12% and 14% respectively) further, however, there still was an extra room for improvement. During this iteration, the early stopping callback that is used to unfreeze the bottom layers and stop the model training was called when the improvement in validation loss was less than 1% for 5 epochs. This number was very low and caused the model to underfit. This happens when the model is not trained for enough amount of time. It can be seen in Figure 6.3 that the model stopped the training after only 19 epochs, which is less than half of the number of epochs trained in the first training iteration. As it can be seen in Figure 6.4, the MAE values were approximately 8 by the end of training.

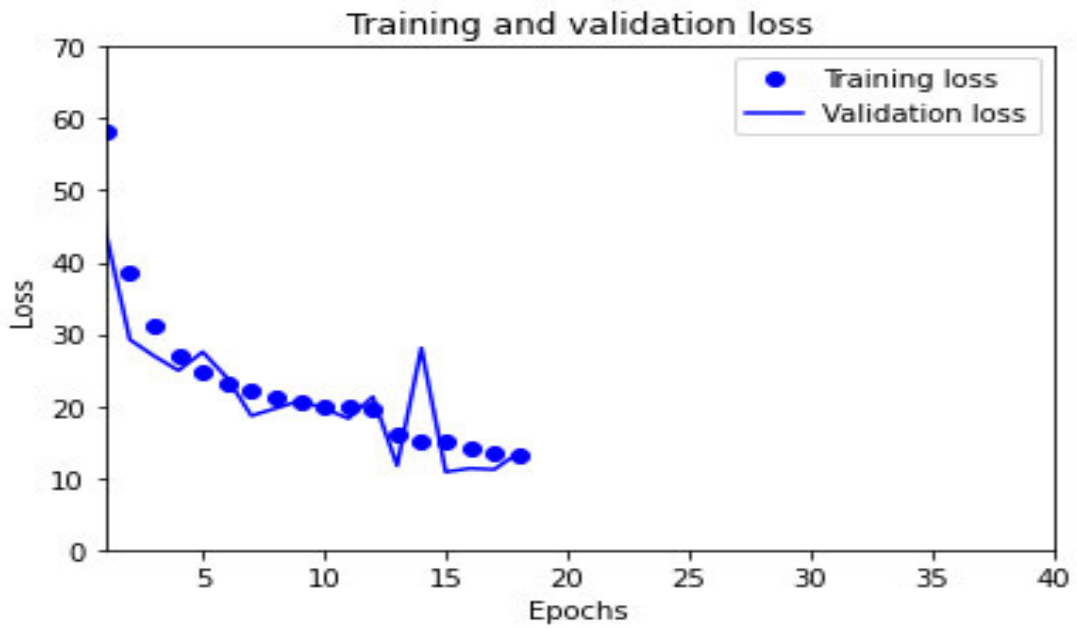


Figure 6.3: Training vs Validation losses during second training iteration.

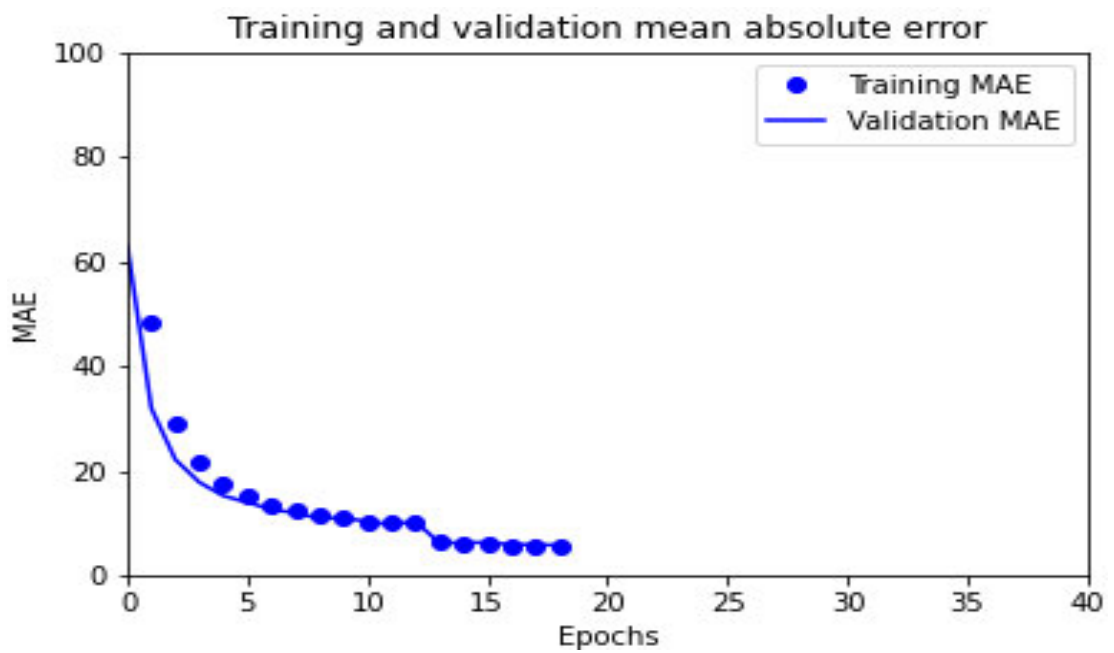


Figure 6.4: MAE values during second training iteration.

Optimizing Early Stopping

For the final tuning step, two different stopping callbacks are defined for the training process. One is defined for when the bottom layers are frozen; training the top layers takes place until the validation loss improvement is less than 0.5% for 10 epochs (see Listing 6.1). After the callback is performed, the bottom layers are unfrozen and the learning rate is decreased by a factor of 0.1 then the model continues training for all the layers which allows for a last step of fine-tuning. The second callback (see Listing 6.2) is called when the improvement in validation loss is less than 1% for 5 epochs. Finally, the training is stopped and the model is saved.

```
1 early_stopping_frozen = EarlyStopping(min_delta=0.5, patience
   =10, monitor="val_loss", verbose=1)
```

Listing 6.1: Early stopping callback when the bottom layers are frozen.

```
1 early_stopping = EarlyStopping(monitor="val_loss", min_delta
   =1, patience=5, verbose=1)
```

Listing 6.2: Early stopping callback when the bottom layers are unfrozen.

Final Outcome

After experimenting with the different parameters, it is deduced that the best outcome is the one achieved during the final tuning step mentioned in the previous section. That iteration produced a model with validation loss of 7% and training loss of 10% by the end of the training (see Figure 6.5). Additionally, it achieved a value of approximately 4 with respect to the MAE (see Figure 6.6).

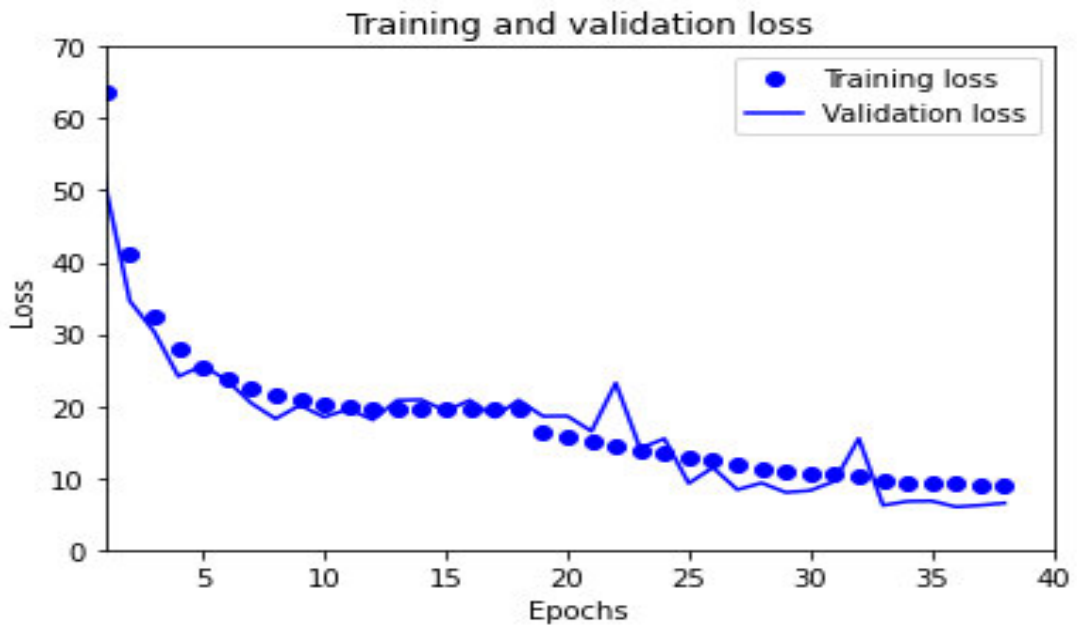


Figure 6.5: Training vs Validation losses during final training iteration.

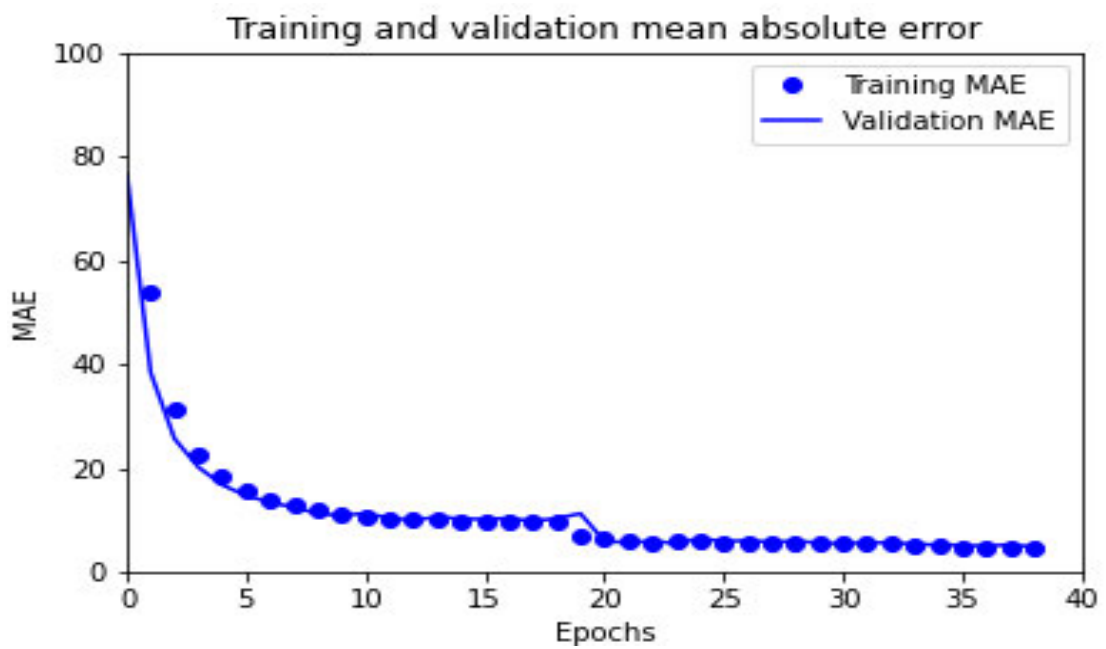


Figure 6.6: MAE values during final training iteration.

The AP is then calculated by running the model on the test dataset and comparing the model predictions to the true values annotated in the beginning of the project. For this purpose, mAP, a python library is used [57]. The final model has a AP@0.5 of approximately 83%. This is about 25% more than the ordinary YOLO model trained on the COCO dataset. The comparison can be seen in Figures 6.7 and 6.8.

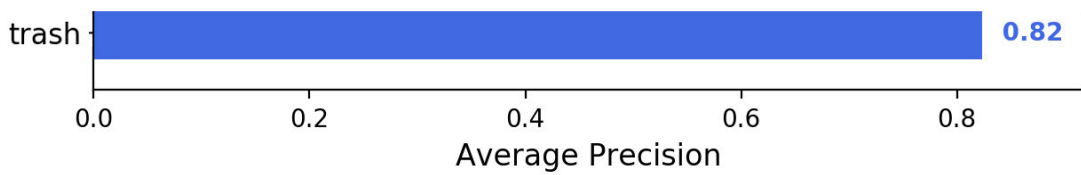


Figure 6.7: AP@0.5 of the final model.

	backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
<i>Two-stage methods</i>							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
<i>One-stage methods</i>							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figure 6.8: mAP of various object detection networks trained on COCO dataset [58].

Despite the fact that this iteration produced a decent model in terms of accuracy, the model was slow and did not meet the requirements of the project. Detection runs at 0.6 FPS on a standard laptop with no GPU. However, the produced model is 10 times faster than the original, unmodified YOLOv3 model. This is due to the fact that the original model can detect objects of 80 different classes and is trained on a much larger dataset. The detection is also run using the same GPU used for training the model

6 Evaluation

which produced a speed of 1.7 FPS. An example of the model predictions on the test dataset can be seen in Figure 6.9.



Figure 6.9: Model predictions on some images of the test dataset.

Tiny YOLO

As the most important metric for this project is the speed of the model, another experimental iteration is done using the Tiny YOLO architecture which can perform detection at a higher speed and lower accuracy. The tiny architecture contains fewer layers and hence has lower weights. This iteration is performed using the same parameters for the final optimized model specified in the previous section. The tiny version has a validation loss of 8% and training loss of 10% (see Figure 6.10).

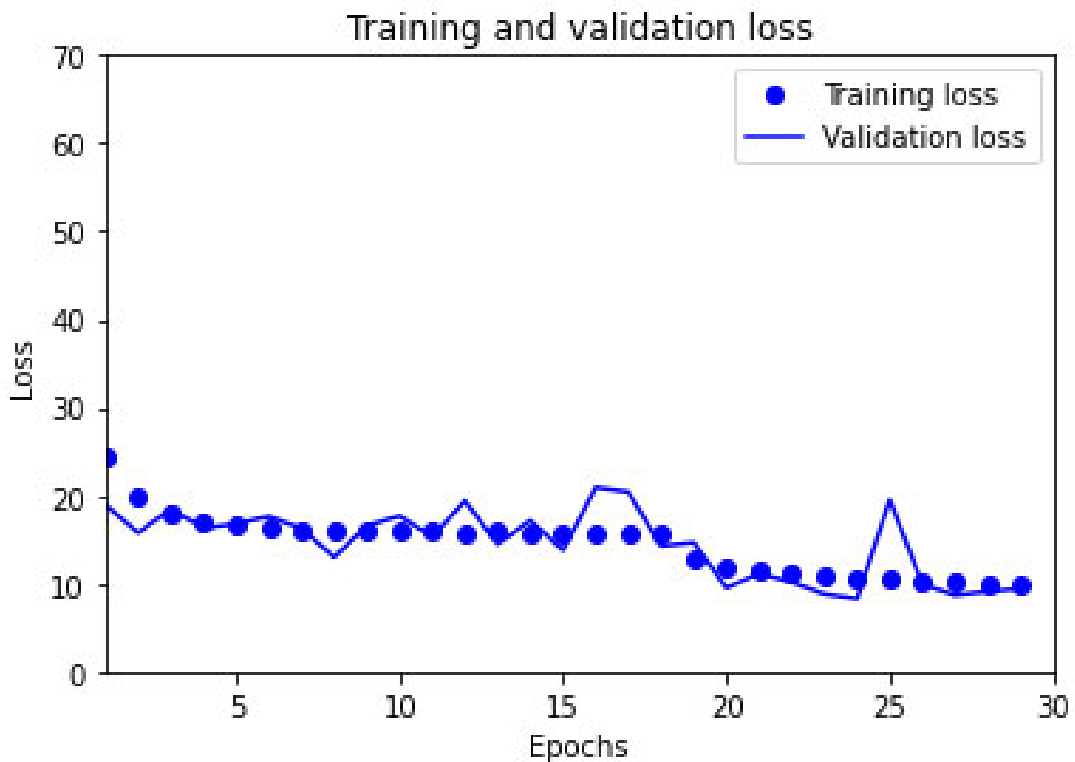


Figure 6.10: Training vs Validation losses for Tiny YOLO architecture.

Additionally, the model is able to run at a speed of 4.5 FPS on a standard laptop without a GPU, and at a speed of 11.4 FPS on the GPU that is used for training, however, the accuracy is lower in this case. On the test data, this model has an AP@0.5 of 64%, which is almost 20% less than the YOLOv3 version (see Figure 6.11). An example of the tiny model predictions on the test dataset can be seen in Figure 6.12.

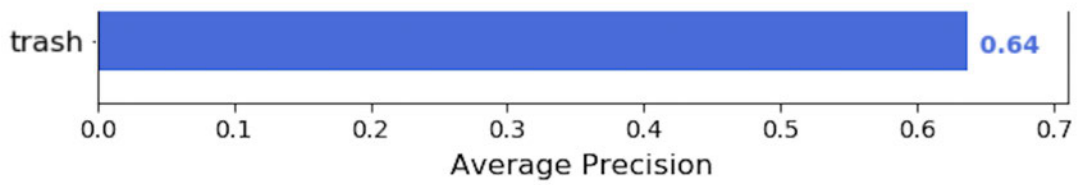


Figure 6.11: AP@0.5 of the tiny model.

6 Evaluation



Figure 6.12: Tiny model predictions on some images of the test dataset.

7 Summary

In this thesis, numerous computer vision techniques are investigated and discussed for the purpose of producing a functional garbage detection model. This chapter summarizes the obtained results, the encountered problems and what future improvements can be done to the work of this project.

7.1 Conclusion

The evaluation of this thesis' outcome is thoroughly discussed in chapter 6, however it is important to stress some conclusions with respect to the requirements of the project. The work of this thesis produced a functional garbage detection model that can operate with high accuracy. Additionally, another model is also produced that is able to perform detection with lower accuracy and at a higher speed.

It has been proven by the work done in this thesis that it is not possible to run a highly accurate, state-of-the-art object detection model that can perform detection at a very high speed on a device with low computing power. There are three factors that influence object detection models which cannot exist simultaneously; accuracy, speed, and computing power. One has to decide which two factors of the three are the most important with respect to the project's requirements.

Even though state-of the art object detection architectures can produce extremely accurate and fast models, they still need to operate on devices with high computing power which cannot be available in some cases.

7.2 Future Work

It is up to successors who are going to continue the work of this project to decide upon which model to use out of the two that were produced. Additionally, data quality can be increased by training the model again on new data that will be collected by the drone. The more relevant data that is fed to the model, the higher its performance can become. Moreover, it is also worthwhile to investigate newer versions of YOLO (YOLOv4 and YOLOv5) and investigate the performance of such architectures.

Appendix

The CD attached to this thesis contains this document in a PDF format, the two produced models in addition to their weights and training histories. The document can be viewed upon application to the principal examiner of this thesis.

Additionally, the source code can be found on this GitHub repository¹ and the complete dataset used for training and testing can be downloaded from here².

¹<https://github.com/moazelsheby/BachelorThesis>

²<https://drive.google.com/file/d/1fznAYf0CHPEPs3OAF3gNRbktchRnSuXP/view?usp=sharing>

Bibliography

- [1] (2020). Production, use, and fate of all plastics ever made, [Online]. Available: <https://advances.sciencemag.org/content/3/7/e1700782> (visited on 07/11/2020) (cit. on p. 1).
- [2] (2020). Waste: A problem or a resource?, [Online]. Available: <https://www.eea.europa.eu/signals/signals-2014/articles/waste-a-problem-or-a-resource> (visited on 07/11/2020) (cit. on p. 1).
- [3] (2020). Municipal waste statistics, [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php/Municipal_waste_statistics (visited on 07/11/2020) (cit. on p. 2).
- [4] Y. Demiral, A. Soysal, A. Bilgin, B. Kilic, B. Unal, R. Uçku, and T. Theorell, “The association of job strain with coronary heart disease and metabolic syndrome in municipal workers in turkey”, *Journal of occupational health*, vol. 48, pp. 332–8, Oct. 2006. DOI: [10.1539/joh.48.332](https://doi.org/10.1539/joh.48.332) (cit. on p. 4).
- [5] (2020). Unesco prizewinner uses ai tool to detect and collect trash worldwide, [Online]. Available: <https://en.unesco.org/news/unesco-prizewinner-uses-ai-tool-detect-and-collect-trash-worldwide> (visited on 07/13/2020) (cit. on p. 4).
- [6] (2020). Let’s do it ai project, [Online]. Available: <https://opendata.letsdoitworld.org/#/ai> (visited on 07/13/2020) (cit. on p. 4).
- [7] (2020). Smart garbage visual detection, monitoring and analytics, [Online]. Available: <https://becominghuman.ai/smart-garbage-visual-detection-monitoring-and-analytics-a0061fff2b76> (visited on 07/13/2020) (cit. on p. 5).

- [8] J. Redmon, *Darknet: Open source neural networks in c*, <http://pjreddie.com/darknet/>, 2013 (cit. on p. 5).
- [9] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement”, *arXiv*, 2018 (cit. on pp. 5, 26).
- [10] (2020). Store and sync data in real time, [Online]. Available: <https://firebase.google.com/products/realtime-database> (visited on 07/13/2020) (cit. on p. 5).
- [11] M. Hossain, B. Debnath, A. Anika, M. J. Al-Hossain, C. Biswas, and Shahnaz, “Autonomous trash collector based on object detection using deep neural network”, Oct. 2019. DOI: [10.1109/TENCON.2019.8929270](https://doi.org/10.1109/TENCON.2019.8929270) (cit. on p. 5).
- [12] (2020). Keras. simple. flexible. powerful., [Online]. Available: <https://keras.io/> (visited on 08/23/2020) (cit. on pp. 5, 30).
- [13] (2020). Opencv, [Online]. Available: <https://opencv.org/> (visited on 08/23/2020) (cit. on pp. 5, 33).
- [14] Y. Liu, Z. Ge, G. Lv, and S. Wang, “Research on automatic garbage detection system based on deep learning and narrowband internet of things”, *Journal of Physics: Conference Series*, vol. 1069, p. 012 032, Aug. 2018. DOI: [10.1088/1742-6596/1069/1/012032](https://doi.org/10.1088/1742-6596/1069/1/012032) (cit. on p. 5).
- [15] F. Chollet, *Deep Learning with Python*, 1st. USA: Manning Publications Co., 2017, ISBN: 1617294438 (cit. on pp. 6, 7, 43).
- [16] C. Molnar, *Interpretable Machine Learning, A Guide for Making Black Box Models Explainable*. 2019, <https://christophm.github.io/interpretable-ml-book/> (cit. on p. 6).
- [17] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012, ISBN: 026201825X (cit. on p. 7).
- [18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org> (cit. on pp. 7, 9).
- [19] J. Heaton, *Introduction to Neural Networks for Java, 2nd Edition*, 2nd. Heaton Research, Inc., 2008, ISBN: 1604390085 (cit. on p. 7).
- [20] J. E. Solem, *Programming Computer Vision with Python*. O’Reilly, 2012, ISBN: 978-144-93-1654-9 (cit. on p. 8).

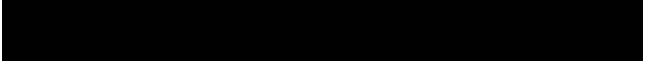
- [21] S. Prince, *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012 (cit. on p. 8).
- [22] X. Jiang, A. Hadid, Y. Pang, E. Granger, and X. Feng, *Deep Learning in Object Detection and Recognition*. Springer Singapore, 2019, ISBN: 978-981-10-5152-4 (cit. on p. 8).
- [23] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2018. [Online]. Available: <http://neuralnetworksanddeeplearning.com/> (cit. on p. 9).
- [24] A. Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd. O’Reilly Media, Inc., 2017, ISBN: 1491962291 (cit. on pp. 10, 11, 39, 43, 48, 49).
- [25] (2020). Convolutional neural networks (lenet), [Online]. Available: <http://deeplearning.net/tutorial/lenet.html> (visited on 09/05/2020) (cit. on p. 10).
- [26] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2014. arXiv: [1411.4038 \[cs.CV\]](https://arxiv.org/abs/1411.4038) (cit. on p. 11).
- [27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2013. arXiv: [1311.2524 \[cs.CV\]](https://arxiv.org/abs/1311.2524) (cit. on p. 11).
- [28] R. Girshick, *Fast r-cnn*, 2015. arXiv: [1504.08083 \[cs.CV\]](https://arxiv.org/abs/1504.08083) (cit. on p. 12).
- [29] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2015. arXiv: [1506.01497 \[cs.CV\]](https://arxiv.org/abs/1506.01497) (cit. on pp. 12, 13).
- [30] (2020). R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithm, [Online]. Available: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e> (visited on 08/01/2020) (cit. on p. 13).
- [31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector”, *Lecture Notes in Computer Science*, 2016, ISSN: 1611-3349. DOI: [10.1007/978-3-319-46448-0_2](https://doi.org/10.1007/978-3-319-46448-0_2). [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2 (cit. on p. 14).

- [32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2015. arXiv: [1506.02640 \[cs.CV\]](https://arxiv.org/abs/1506.02640) (cit. on pp. 15, 16).
- [33] (2020). Yolo v5 is here! custom object detection tutorial with yolo v5, [Online]. Available: <https://medium.com/towards-artificial-intelligence/yolo-v5-is-here-custom-object-detection-tutorial-with-yolo-v5-12666ee1774e> (visited on 09/02/2020) (cit. on pp. 15, 18).
- [34] J. Redmon and A. Farhadi, *Yolo9000: Better, faster, stronger*, 2016. arXiv: [1612.08242 \[cs.CV\]](https://arxiv.org/abs/1612.08242) (cit. on p. 16).
- [35] —, *Yolov3: An incremental improvement*, 2018. arXiv: [1804.02767 \[cs.CV\]](https://arxiv.org/abs/1804.02767) (cit. on p. 16).
- [36] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2017. arXiv: [1708.02002 \[cs.CV\]](https://arxiv.org/abs/1708.02002) (cit. on p. 17).
- [37] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: [2004.10934 \[cs.CV\]](https://arxiv.org/abs/2004.10934) (cit. on pp. 17, 18).
- [38] M. Tan, R. Pang, and Q. V. Le, *Efficientdet: Scalable and efficient object detection*, 2019. arXiv: [1911.09070 \[cs.CV\]](https://arxiv.org/abs/1911.09070) (cit. on p. 17).
- [39] G. Jocher, A. Stoken, and J. Borovec, *Ultralytics/yolov5: V3.0*, version v3.0, Aug. 2020. DOI: [10.5281/zenodo.3983579](https://doi.org/10.5281/zenodo.3983579). [Online]. Available: <https://doi.org/10.5281/zenodo.3983579> (cit. on pp. 18, 19).
- [40] (2020). Machine learning life cycle, [Online]. Available: <https://www.educba.com/machine-learning-life-cycle/> (visited on 08/31/2020) (cit. on p. 21).
- [41] (2020). Solving data challenges in machine learning with automated tools, [Online]. Available: <https://www.topbots.com/data-preparation-for-machine-learning/> (visited on 08/31/2020) (cit. on p. 21).
- [42] J. Ruiz-del-Solar, P. Loncomilla, and C. Soto, “A survey on deep learning methods for robot vision”, Mar. 2018 (cit. on p. 24).

- [43] J. Redmon. (2020). Yolo: Real-time object detection, [Online]. Available: <https://pjreddie.com/darknet/yolo/> (visited on 08/17/2020) (cit. on p. 28).
- [44] (2020). Coco common objects in context, [Online]. Available: <https://cocodataset.org/#home> (visited on 08/17/2020) (cit. on p. 28).
- [45] (2020). The state of the octoverse: Machine learning, [Online]. Available: <https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/> (visited on 08/22/2020) (cit. on p. 29).
- [46] (2020). An end-to-end open source machine learning platform, [Online]. Available: <https://www.tensorflow.org/> (visited on 08/23/2020) (cit. on p. 30).
- [47] (2020). Pytorch from research to production, [Online]. Available: <https://pytorch.org/> (visited on 09/06/2020) (cit. on p. 31).
- [48] (2020). Matlab for machine learning, [Online]. Available: <https://www.mathworks.com/solutions/machine-learning.html> (visited on 08/23/2020) (cit. on p. 33).
- [49] (2020). Imagej an open platform for scientific image analysis, [Online]. Available: <https://imagej.net/Welcome> (visited on 09/06/2020) (cit. on p. 33).
- [50] G. Thung and M. Yang, *TrainyourownYOLO: Building a custom object detector from scratch*, 2020. [Online]. Available: <https://github.com/garythung/trashnet> (cit. on pp. 36, 37).
- [51] *Ai tool wade*, 2020. [Online]. Available: <https://github.com/letsdoitworld/wade-ai> (cit. on pp. 36, 38).
- [52] (2020). Python-resize-image 1.1.19, [Online]. Available: <https://pypi.org/project/python-resize-image/> (visited on 08/24/2020) (cit. on p. 36).
- [53] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernández, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, *et al.*, *Imgaug*, 2020. [Online]. Available: <https://github.com/aleju/imgaug> (cit. on p. 39).

- [54] (2020). What is data annotation and what are its advantages?, [Online]. Available: <https://medium.com/analytics/what-is-data-annotation-and-what-are-its-advantages-95766213351e> (visited on 09/11/2020) (cit. on p. 41).
- [55] (2020). Visual object tagging tool, [Online]. Available: <https://github.com/microsoft/VoTT> (visited on 09/11/2020) (cit. on p. 42).
- [56] A. Muehlemann, *Trainyourownyolo: Building a custom object detector from scratch*, 2019. [Online]. Available: <https://github.com/AntonMu/TrainYourOwnYOLO> (cit. on p. 42).
- [57] J. Cartucho, R. Ventura, and M. Veloso, “Robust object recognition through symbiotic deep learning in mobile robots”, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2336–2341 (cit. on p. 54).
- [58] (2020). Review: Yolov3 — you only look once (object detection), [Online]. Available: <https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6> (visited on 09/14/2020) (cit. on p. 54).

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. October 2020 
Moaz Elshebly