ENHANCING TURKISH TRAFFIC SIGN RECOGNITION: A COMPARISON OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

by

Kaan Kocakanat

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Computer Engineering

Yeditepe University
2022

ENHANCING TURKISH TRAFFIC SIGN RECOGNITION: A COMPARISON OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

APPROVED BY:

Assist. Prof. Dr. Tacha Serif .....
(Thesis Supervisor)
(Yeditepe University)

Prof. Dr. Sezer Gören Uğurdağ .....
(Yeditepe University)

Prof. Dr. Cem Ünsalan .....
(Marmara University)

DATE OF APPROVAL: ..../..../2022

I hereby declare that this thesis is my own work and that all information in this thesis has been obtained and presented in accordance with academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. If any material used in the thesis requires copyright, the necessary permissions have been obtained. No material from this thesis has been used for the award of another degree.

I accept all kinds of legal liability that may arise in case contrary to these situations.

Name, Last name Kaan Kocakanat

Signature …..……..……

ACKNOWLEDGEMENTS

I'd like to show my thankfulness to Dr. Tacha Serif, my project supervisor, for all of his advice and assistance. Working on my thesis under his supervision has always been a motivating and unique experience for me, allowing me to learn about a variety of subjects. I'd also like to show my appreciation to him for motivating me to collaborate with him on the publishing of my first academic paper.

Finally, I'd like to show my heartfelt gratitude to my family for supporting me unconditionally. I was successful in completing this project with their help and emotional support. I am grateful to my parents for teaching me that a task should never be left until it is done.

ABSTRACT

ENHANCING TURKISH TRAFFIC SIGN RECOGNITION: A COMPARISON OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

As the number of vehicles on the roads increases, traffic signs are becoming more and more important every passing day. Despite the fact that traffic signs are simple and easy to understand, in congested traffic drivers may miss them. It would be a big help if a system could assist the driver with traffic signs. A traffic sign recognition (TSR) system needs to be implemented to achieve this. Three different steps are proposed in this thesis for the Turkish TSR system. Firstly, as step one, models are trained using selective search method and transfer learning is utilized in model training. Models are developed using the pre-trained MobileNetV2 model. Secondly, as step two, models are trained using the selective search method once more, however this time instead of using the transfer learning, the models are trained by developing a new custom CNN. Lastly, as step three, the Faster R-CNN Inception V2 COCO model is utilized in model training. For training purposes, indigenous dataset is created containing 54 distinct classes and 10842 Turkish traffic sign images. The training process of the models are carried out twice with different training step numbers. Then, these models are used to detect Turkish traffic sign images taken both daytime and nighttime with large size and small size images. The results of step one indicate that the loss value for the model which is trained with 25 epochs is 7 percent, with a test accuracy of 98.7 percent and the loss value for the model which is trained with 50 epochs is 4 percent, with a test accuracy of 99 percent. The results of step two show that the loss value for the model which is trained with 25 epochs is 8.9 percent, with a test accuracy of 98.8 percent and the loss value for the model which is trained with 50 epochs is 9.5 percent, with a test accuracy of 99.1 percent. The results of step three demonstrates that the Faster R-CNN model's average precision is 67.2 percent and average recall is 78.3 percent when trained with 51,217 steps; on the other hand, the average precision increases to 76 percent and average recall increases to 82.8 percent when trained with 200,000 steps. When the total test time for all three steps are compared, the detection and recognition time for the models are approximately 51, 63, and 7 seconds, respectively.

ÖZET

TÜRK TRAFİK İŞARETİ TANIMASININ GELİŞTİRİLMESİ: EĞİTİM ADIM SAYILARI, AYDINLATMA KOŞULLARI VE GÖRÜNTÜ BOYUTLARININ KARŞILAŞTIRILMASI

Yollardaki araç sayısının her geçen gün artmasıyla birlikte trafik işaretleri her geçen gün daha da önem kazanmaktadır. Trafik işaretleri basit ve anlaşılması kolay olmasına rağmen, sıkışık trafikte sürücüler bunları gözden kaçırabilir. Sürücüye trafik işaretleri konusunda yardımcı olması için bir trafik işareti tanıma sisteminin uygulanması gerekmektedir. Bu tezde Türk trafik işareti tanıma sistemi için üç farklı adım önerilmiştir. İlk olarak, birinci adımda, modeller seçici arama algoritması kullanılarak eğitilir ve model eğitiminde öğrenme aktarımı kullanılır. Modeller, önceden eğitilmiş MobileNetV2 modeli kullanılarak geliştirilmiştir. İkinci olarak, ikinci adımda, modeller bir kez daha seçici arama algoritması kullanılarak eğitilir, ancak bu sefer öğrenme aktarımını kullanmak yerine, modeller yeni geliştirilen bir CNN ile eğitilir. Son olarak, üçüncü adımda, model eğitiminde Daha Hızlı R-CNN Inception V2 COCO modeli kullanılır. Modellerin eğitimi sırasında kullanılması amacıyla 54 farklı sınıf ve 10842 adet Türk trafik işareti görüntüsünü içeren özgün veri seti oluşturulmuştur. Modellerin eğitim süreci, farklı eğitim adım sayıları ile iki kez gerçekleştirilir. Daha sonra bu modeller, hem gündüz hem de gece çekilen Türk trafik işareti görüntülerini büyük ve küçük boyutlu görüntülerle tespit etmek için kullanılır. Birinci adımın sonuçları, 25 epok ile eğitilen model için kayıp değerinin yüzde 7 ve test doğruluğunun yüzde 98.7 olduğunu gösterirken 50 epok ile eğitilen model için kayıp değerinin yüzde 4 ve test doğruluğunun yüzde 99 olduğunu göstermektedir. İkinci adımın sonuçları, 25 epok ile eğitilen model için kayıp değerinin yüzde 8.9 ve test doğruluğunun yüzde 98.8 olduğunu gösterirken 50 epok ile eğitilen model için kayıp değerinin yüzde 9.5 ve test doğruluğunun yüzde 99.1 olduğunu göstermektedir. Üçüncü adımın sonuçları, 51,217 adımla eğitildiğinde Daha Hızlı R-CNN modelinin ortalama hassasiyetinin yüzde 67.2 ve ortalama hatırlamanın yüzde 78.3 olduğunu, 200,000 adımla eğitildiğinde ise ortalama hassasiyetinin yüzde 76'ya ve ortalama hatırlamanın yüzde 82.8'e yükseldiğini göstermektedir. Her üç adım için toplam test süresi karşılaştırıldığında, modeller için algılama ve tanıma süresinin sırasıyla yaklaşık 51, 63 ve 7 saniye olduğu görülmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

TSR	Traffic sign recognition
R-CNN	Region-based convolutional neural network
MS COCO	Microsoft common objects in context
CNN	Convolutional neural network
AI	Artificial intelligence
OCR	Optical character recognition
ICR	Intelligent character recognition
ILSVRC	ImageNet large scale visual recognition challenge
SVM	Support vector machine
HSI	Hue, saturation, intensity
ANN	Artificial neural network
RoI	Region of interest
KAIST	Korea advanced institute of science and technology
GTSDB	German traffic sign detection benchmark
GTSRB	German traffic sign recognition benchmark
YOLO	You only look once
GPU	Graphics processing unit
CPU	Central processing unit
TPU	Tensor processing unit
VOC	Visual object classes
SPP	Spatial pyramid pooling
mAP	Mean average precision
VGG	Visual geometry group
C-CNN	Color segmentation - convolutional neural network
API	Application programming interface
AP	Average precision
AR	Average recall
RPN	Region proposal network
SSD	Single shot detector
AN	Attention network

TT100K	Tsinghua-Tencent 100K
BTSD	Belgium traffic signs dataset
RGB	Red, green, blue
MNIST	Modified national institute of standards and technology
ReLU	Rectified linear unit
NMS	Non-Maximum Suppression
BBR	Bounding box regression
DPM	Deformable part models
PNG	Portable network graphics
JPEG	Joint photographic experts group
IoU	Intersection over union
TP	True positive
FP	False positive
FN	False negative
XML	Extensible markup language
CSV	Comma-separated values

1. INTRODUCTION

In the following chapter, firstly, the definition of the problem is described in section 1.1, then, the motivation is explained in section 1.2, and lastly, the thesis' structure is shown in section 1.3.

1.1. PROBLEM DEFINITION

Undeniably, the number of vehicles is rapidly increasing globally. In Turkey, it is increasing at the same pace as in the world. Information about vehicles in the last ten years in Turkey is shown in Table 1.1. The number of motor vehicles in Turkey has increased by nearly 60 percent during the previous ten years [1]. As the number of vehicles on the road increases from year to year, more motor vehicles are expected to be on the road in the coming years.

Table 1.1. Number of road motor vehicles [1]

Year	Total	Car	Minibus	Bus	Small truck	Truck	Motorcycle	Special Vehicles	Tractor
2011	**16 089 528**	8 113 111	389 435	219 906	2 611 104	728 458	2 527 190	34 116	1 466 208
2012	**17 033 413**	8 648 875	396 119	235 949	2 794 606	751 650	2 657 722	33 071	1 515 421
2013	**17 939 447**	9 283 923	421 848	219 885	2 933 050	755 950	2 722 826	36 148	1 565 817
2014	**18 828 721**	9 857 915	427 264	211 200	3 062 479	773 728	2 828 466	40 731	1 626 938
2015	**19 994 472**	10 589 337	449 213	217 056	3 255 299	804 319	2 938 364	45 732	1 695 152
2016	**21 090 424**	11 317 998	463 933	220 361	3 442 483	825 334	3 003 733	50 818	1 765 764
2017	**22 218 945**	12 035 978	478 618	221 885	3 642 625	838 718	3 102 800	60 099	1 838 222
2018	**22 865 921**	12 398 190	487 527	218 523	3 755 580	845 462	3 211 328	63 359	1 885 952
2019	**23 156 975**	12 503 049	493 373	213 358	3 796 919	844 481	3 331 326	65 470	1 908 999
2020	**24 144 857**	13 099 041	493 395	212 407	3 938 732	859 670	3 512 576	70 309	1 958 727

As the number of vehicles in traffic increases, so does the level of uncertainty in the traffic environment. To eliminate this uncertainty and manage traffic flow, traffic signs are placed on the roads. They are designed to be easy to understand so that drivers are aware of the present condition and other crucial information on the road. However, accidents might still happen if drivers fail to pay attention to traffic signals [2]. Table 1.2. illustrates that the driver is to blame for the majority of the accidents that occurred in Turkey in 2020. The fact that drivers are responsible for 157,128 of the 177,867 accidents emphasizes the

importance of this issue. For this very reason, a Traffic Sign Recognition (TSR) system needs to be implemented in order to prevent accidents.

Table 1.2. Faults causing traffic accidents involving death or injury in Turkey, 2020 [3]

Faults	Number of Accidents
Driver faults	157,128
Passenger faults	2,577
Pedestrian faults	12,520
Road faults	897
Vehicle faults	4,745
Total	177,867

1.2. MOTIVATION

A TSR system should be introduced to minimize the risk in traffic caused by drivers and to improve the safety of the traffic environment. TSR is a basic but incredibly important technique that will solve many of the problems that many drivers have in traffic. Even though there are many studies on TSR systems, very few of these studies evaluate real-life driving conditions, such as small size traffic signs or low lighting conditions - such as nighttime. Furthermore, designing a TSR system for a specific country is challenging because of a lack of public datasets - i.e even though multiple TSR studies were undertaken for various countries in the literature, most of them have based their work on the German traffic sign datasets [4], [5]. In addition, while traffic signs in most countries may seem to be similar, there are actually major differences between country traffic signs even within the same economical regions - e.g. Europe, United Kingdom, Americas, Asia and Australia. Accordingly, the aim of this thesis is to develop a prototype Turkish TSR system using a newly created Turkish Traffic Sign Dataset and to investigate the effect of training step numbers, lighting conditions, and image sizes on model performance. Therefore, as part of the thesis a Turkish Traffic Sign Dataset is created. Turkish traffic

sign images are collected to train the models, which are obtained from Google maps. Firstly, as step one, models are trained using the selective search method. In step one, transfer learning is utilized in model training and models are developed using the pre-trained MobileNetV2 model. Secondly, as step two, models are trained using the selective search method once more, however this time instead of using the transfer learning, the models are trained by developing a new custom CNN. Although the accuracy rate is high, the time in detection and recognition in step one and step two during evaluation is pretty long. Because of the use of selective search, the majority of the time is spent generating proposals. For this reason, in order to be able to recognize and detect both with high accuracy and faster, as step three, the Faster R-CNN Inception V2 COCO model is utilized in model training. During the evaluation of the models after the training process, it is observed that traffic signs are detected and recognized faster with higher accuracy with the Faster R-CNN.

1.3. THESIS STRUCTURE

Accordingly, the following is an overview of the thesis' structure; Chapter 2 discusses the previous work undertaken in TSR, Chapter 3 reviews and details the methods used for TSR, Chapter 4 describes the creation and design of the Turkish Traffic Sign Dataset, Chapter 5 explains the implementation and evaluation of the model trained with selective search method and transfer learning approach using the Turkish Traffic Sign Dataset, Chapter 6 presents the development and testing of the model trained with selective search method and newly created custom CNN using the Turkish Traffic Sign Dataset, Chapter 7 introduces the implementation and evaluation of the model trained with the Faster R-CNN Inception V2 COCO model using the Turkish Traffic Sign Dataset, Chapter 8 provides an overview of the results of the three different approaches and information on comparing models. Last but not least, Chapter 9 draws the conclusion and discusses possible future venues for improvement.

2. BACKGROUND

This chapter explores previous studies on traffic signs recognition and elaborates on each one of the studies by grouping them according to their algorithms. First, the evolution of computer vision is discussed in section 2.1 followed by previous studies based on traditional traffic sign recognition methods in section 2.2 and section 2.3 discusses previous studies based on the deep learning based approaches include R-CNN, Fast R-CNN and Faster R-CNN

2.1. EVOLUTION OF COMPUTER VISION

For many years, people dreamed of creating intelligent machines that could think and act like humans. Giving them the ability to see and interpret their surroundings was one of the most fascinating ideas. With the development of technology, these dreams have gradually become reality today as computer vision technology has become more and more involved in our daily lives. Computer vision (CV) is a crucial branch of artificial intelligence (AI) science. [6]. It is the clear and meaningful extraction of objects from images. The aim of computer vision is understanding and interpreting the visual world by processing data with computers in order to obtain meaningful information. With computer vision, it is ensured that the decisions or actions that a person can take with information based on visuals are performed by computers. Although human vision and computer vision work very similarly, human vision has great advantages in recognizing objects, distinguishing them from each other and determining their properties. This advantage comes from the fact that humans grow up doing this naturally. The act of seeing is an ordinary task that is always done, like breathing. At the same time, computer vision needs to process a massive amount of data by using a vast range of algorithms to perform these functions.

In retrospect, studies based on the fundamentals of computer vision began a long time ago. The first studies date back to the late 1950s and early 1960s. In 1959, Hubel and Wiesel [7] broadened the knowledge of our visual system understanding. This study showed neurons in the visual cortex are arranged in a hierarchical architecture. The study conducted on cats showed that neurons detect simple features such as edges. Then these neurons fed the upper part of the brain to detect complex features like shapes. Eventually, the information

was transmitted to a higher area of the brain to form complex visual representations. This experiment was a pioneer in understanding how the visual system developed. In the following years, studies were carried out for computers to recognize objects. Building a system that can analyze an environment and identify objects in the environment was considered a simple step towards solving more complex problems by some of the early pioneers of AI and Robotics. In 1966, Papert and Minsky [8] asked an undergraduate student Gerald Jay Sussman to build a computer system to recognize objects as a summer project. However, this process was done manually. Different rules were written for each object and the machine was asked to guess what the object was. It was difficult to write manual rules for all objects due to the fact that each object is at different angles and different light. The project was not successful. However, it was, according to many, since it is attributed with helping to establish CV as a field of science. About a decade later, In 1974, optical character recognition (OCR) technology which is widely used in different fields today was introduced. Before 1974, readable fonts had to be specially designed to be machine readable only. The first multi-font OCR software is developed by Kurzweil Technologies which is able to recognize text printed in virtually any font. Similarly, neural networks are utilized to recognize text handwritten and block letters using intelligent character recognition (ICR), which is an advanced OCR. Since then, OCR and ICR technologies have been used in applications such as license plate recognition, mobile payments, and machine translation. By the end of the 1970s, Fukushima [9] proposed a novel self-organizing network model which is called Neocognitron. Hubel and Wiesel's work on cells that are simple and complicated in 1962 influenced this study. This network model has a multi-layer structure which includes convolutional layers and has been proposed for recognizing patterns. The proposed network model is able to learn objects; however, similar to its inspiring project, the model in this study did not yield good results either. Nevertheless, it represents a significant advancement in the field of CV. By the early 1980s, Yann LeCun et al. [10] introduced the first study on modern convolutional neural networks (CNNs), inspired by the Neocognitron which is a very basic image recognition neural network. In this study, a backpropagation learning algorithm is applied to Fukushima's CNN architecture. Yann LeCun et al. presented LeNet-5, the first successful CNN model in the 1990s. This novel model did produce good results in recognizing handwritten digits, checks and postal numbers. However, despite the good results, the LeNet-5 model faced problems because it is not scalable. At that time, the technique could

only be applied to low resolution images. In order for the CNN model to yield better results, large data sets containing more images were needed. The ImageNet Large Scale Visual Recognition Competition (ILSVRC) [11] data set was released in 2010. It contains more than one million manually labeled natural images in a thousand object classes. The ILSVRC is a competition for image classification which runs annually. In this competition, research teams assess their algorithms in a particular dataset, then compete for the highest accuracy in different visual recognition tasks. The error rate for ILSVRC winners was approximately 26 percent in 2010 and 2011. In 2012, Alex Krizhevsky et al. [12] introduced a CNN model which is architecturally close to Yann LeCun's LeNet-5. That specific CNN model is then renamed as AlexNet, after winning the ILSVRC. The AlexNet model reached a 16.4 percent error rate. For CNNs, it was a watershed moment. In the subsequent years, the ILSVRC's error rate has dropped to less than 5 percent. When it comes to today, the world is going through a deep digital transformation. Despite the work done so far with the development of technology, we are not yet close to figuring out computer vision. Therefore, there are still numerous issues to be resolved. With the significant progress made in the recent decade, computer vision continues to be a hot topic.

Computer vision is now present in practically all of the technology we use in our daily life. Although we are not aware of it, most applications we use on a daily basis make use of are based on computer vision. For instance, like showing our face to the phone to unlock it. There are many different areas where computer vision applications are used, such as healthcare, security, entertainment, manufacturing and autonomous vehicles. Autonomous vehicles can be considered for the most popular usage area of computer vision applications. Computer vision has been an important part of advances in autonomous vehicles. The increasing number of vehicles in traffic due to the increasing population, causes an increase in the uncertainty in the traffic environment. Traffic signs greatly reduce uncertainties in the traffic environment and minimize dangerous situations that may occur in traffic. However, accidents might still happen if drivers fail to pay attention to traffic signals. The purpose of autonomous vehicles is to keep both drivers and pedestrians safer on the roads. For this reason, it is very important for autonomous vehicles to automatically recognize and detect signs in a high traffic environment. CV is used in the development of autonomous vehicles to interpret visual data from the car's cameras and other sensors. It's critical to be able to recognize traffic signs on the road. To do this, there are several studies

in the literature concerned with TSR systems. If we broadly examine the literature, the studies in the field of TSR can be divided into two groups. Traditional TSR systems are the first group, whereas deep learning-based TSR systems are the second.

2.2. TRADITIONAL TSR METHODS

Traditional TSR methods consist of models that include techniques developed to recognize characteristic features of traffic signs. For example, its color and shape. One of these techniques is color segmentation. It is a method of segmenting an image into different portions based on its colors. Maldonado-Bascón et al. [13] developed a TSR system which is using Support Vector Machines (SVM) in 2007. This study is implemented to be used in driver support systems in the future. The proposed system is designed to cover all current Spanish traffic sign shapes by recognizing different geometric signs such as circular, triangular etc. The proposed system consists of three steps. The first one is segmentation by pixel color, the second one is shape classification with linear SVMs used to detect traffic signs and the third one is using Gaussian-kernel SVMs to recognize content. To obtain thresholds, a histogram of hue and saturation of manually segmented signs is built during the segmentation step. Possible traffic signs are obtained when the segmentation procedure is done. Then, linear SVMs classify them based on their shape. In the training process, it is stated that a library for SVM is used. The proposed system is tested by mounting a camera on the windshield of the vehicle and cruising at a constant speed. The final recognition step had a high success rate, according to the results of the experiment. The methods mentioned above can improve performance to a point, but some factors can cause this performance to decrease, for example; traffic signs are likely to fade and the traffic sign cannot be detected because the sign and the background are so identical. For these reasons, successful detections can be made with shape-based techniques. These techniques make use of the traffic signs invariance and symmetry. This makes it more robust against changes in lighting. For detecting shapes such as circles and triangles, one of the most commonly used shape-based approaches is the Hough transform. Garcia-Garrido et al. [14] implemented a TSR system as a solution to changing lighting conditions in 2006. The main reason for this study is due to the fact that there were many studies on TSR systems but these studies rarely yield results at night or under adverse weather conditions. The proposed system utilized the Hough transform method for sign detection. After the

detection stage, for recognition, two distinct neural networks are used. The networks are enabled to recognize circular and triangular signs. The test is done in real time and the classification rate is reached 99 percent. As seen in Figure 2.1, the proposed system has been tested with a vehicle mounted camera under different lighting conditions and a variety of weather situations, such as, sunny, rainy etc.

Figure 2.1. One camera mounted on the windscreen of the car [14]

The results show that the average processing time is approximately 30ms. However, when the Hough transform is extended to ellipses, the processing time drops to approximately 2 seconds. Due to this fact, the proposed system is not a good candidate for a real-time operation. Although these methods are very good at classification, they are time-consuming. Of course, there are studies aiming to decrease the processing time by technologies such as radial symmetry detector. In 2008, an example to this is the work by Barnes et al. [15] who introduced a radial symmetry detector for detecting signs. The main focus of the study is designing a system that could help drivers straight away. In this study, using the fast radial symmetry detector, an effective approach for speedy sign detection is examined. Hough circle detection which is a powerful approach takes a lot of time to process large images on the other hand, fast radial symmetry processes in less time. The proposed system is tested in real time on the Australian National University's intelligent vehicle. According to the results, the proposed approach has 93 percent successful detection. In other words, it can detect signs in real time with high accuracy. However, the proposed system may have difficulties in recognizing traffic signs as it is sensitive to noise in dispersed environments. Since the methods mentioned previously work on the whole image, the number of regions to be considered by the algorithm is too high. Therefore,

there are still some difficulties in using these methods to recognize and detect traffic signs in real time.

2.3. DEEP LEARNING-BASED APPROACHES

Multilayer Artificial Neural Networks (ANNs) are referred to as Deep Learning. Principally, the approaches using deep learning learn the general features with the help of data sets. CNN is one of the most widely used deep learning techniques [16]. Today, CNN is considered a powerful tool for a lot of applications and it has been widely used for TSR problems. Unlike the traditional methods, this approach suggests executing the algorithm on equally divided parts of the same image rather than the entire image. Jung et al. [17] conducted a study to recognize traffic signs in 2016. The aim of the study is to design a vision-based TSR system that would alert the driver about the speed limits. A CNN-based TSR system is implemented utilizing the Hough transform algorithm for extracting candidate regions of traffic signs.

Figure 2.2. Example of Korean-version traffic signs [17]

The LeNet-5 model is used to build deep CNNs and The Caffe framework is used to execute efficient computer operations. The proposed TSR system has been developed by C++, CUDA 7.5 environment. A dataset is created by collecting Korean traffic sign images

for the training of Korean version traffic signs and the dataset includes 16 traffic signs of 6 different types as visualized in Fig 2.2. The suggested system is tested on the KAIST campus road. According to the results of the tests, all traffic signs are accurately recognized. The authors, based on their observation, state that the model could learn the data in a consistent way, and its accuracy will improve as more data is added. Another TSR study is undertaken by Chaudhari et al. [18] in 2020. The purpose of the study presented is to learn small-scale CNN faster with higher accuracy. The proposed solution utilized an approach that can be applied to different applications to recognize traffic signs using CNN on a small scale. In the proposed CNN model, categorical cross entropy is used. The model code is written in Python and Keras API is implemented. This proposed model is trained using the German Traffic Sign Recognition Benchmark (GTSRB) dataset. It contains 43 different types of traffic signs with more than 50K images. Google-colab is used for all the experiments. The training process of the model is carried out with 50 epochs and the proposed CNN model is applied to an advanced driver-assistance systems based system using GTSRB dataset. The results showed that testing accuracy is 97.71 percent, training accuracy is 99.19 percent and validation accuracy is 99.61 percent. The results indicate that the proposed model gives an important improvement in testing accuracy compared with LeNet deep learning models. Another remarkable study on the TSR system in 2020 is conducted by Novak et al. [19]. The You Only Look Once (YOLO) network is used in this study, as it is necessary to recognize traffic signs around vehicles in real time for safe driving. This is a special case where the proposed system is built on YOLO v3 and enhanced with a CNN. While the YOLO v3 model is used to detect participants in the traffic, the CNN model is utilized to classify them. Keras is used as a deep learning framework. In this study, Belgium Traffic Signs Dataset and The GTSRB are combined. The combined dataset contains around 120.000 images of traffic signs which is divided into 75 categories. The proposed system is tested in a wide range of driving environments and the recognition accuracy is more than 99.2 percent of examined signs. The proposed system which is built on YOLO v3 and enhanced with a CNN provides a better traffic sign classification solution, on the other hand this algorithm works slower.

2.3.1. Region-Based Convolutional Neural Networks (R-CNN) Approaches

Region-based convolutional neural networks (R-CNN) is a machine learning model and it is used especially in object detection studies. The main purpose of the R-CNN approach is to use a picture as input and generate a group of bounding boxes as output. The object and the category are both included inside these bounding boxes. R-CNN has been used to track objects, for example traffic sign recognition. In this section, R-CNN algorithms in the literature are discussed. Each of these algorithms attempts to optimize, speed up or enhance object recognition results. Girshick et al. [20] introduced a novel method for all modern object detection networks. This study is unique due to combining region proposals with CNN's. This new method is called R-CNN. It is developed to find objects in an image. The system has received an image as input and extracted approximately 2000 region proposals. To generate region proposals, the selective search method is employed. Then, using a massive CNN, the suggested system generated features for each region proposal. Linear SVMs are used to classify each region, regardless of its shape. Each region is combined, and if there is a regional overlap, the proposal that obtained the highest score calculated by the SVM is taken. This step is done independently for each object class. Thus, only regions with points higher than 0.5 are kept. The test is conducted on the PASCAL VOC dataset and the results on PASCAL VOC 2010-12 showed that the proposed system achieved an improvement from 35.1 percent to 53.7 percent mAP. An impressive improvement in object detection accuracy has been achieved with the R-CNN method. However, with this approach, while the training process takes a lot of time, it also takes up a lot of space in terms of storage. Also, object detection is slow. R-CNN with VGG-16 takes 47 seconds per image during testing which makes it an unlikely solution in order to detect objects in real time. The R-CNN method is slow since there is no shared computing in the whole process. He et al. [21] presented a new network to increase the R-CNN by sharing computation. The fixed size images may reduce the recognition accuracy in previously deep CNNs. They are usually generated through crop and warp. For example, a result of cropping the image, the entire object may not be obtained from the image. To solve this problem, spatial pyramid pooling (SPP-net), is introduced in this study. In SPP-net, a SSP layer is placed between the top of the last convolutional layer and fully connected to remove the fixed-size constraint of the network. In the SPP-net method, the feature map is computed only once for the whole input image. Then, using a feature

vector taken from the shared feature map, each object proposal is categorized. During the training process, two different training methods were applied: single size training and multi size training. The proposed method is evaluated on the Pascal VOC dataset. The results show that the SPP-net has a 44.9 percent accuracy, which is greater than the 44.2 percent accuracy of the R-CNN. Although it has comparable accuracy, the SPP-net is much better than R-CNN in terms of speed. At test time, SPP-net requires 2.3 seconds per image for detection while R-CNN requires 47 seconds.

2.3.2. Fast R-CNN Approaches

The SPP-net method is faster than R-CNN, however it has notable drawbacks. Although the SPP layer prevents re-extraction of features for different regions, the training process consists of a multi-stage pipeline which includes region proposal, CNN feature extraction and SVM classification. Moreover, it is costly in terms of storage as feature vectors are also stored on disk. Girshick [22] who is the author of the R-CNN paper proposed a new approach for object detection to address the issues raised above. The Fast Region-based Convolutional Network model (Fast R-CNN) is introduced in this study. Instead of using a multi-stage pipeline, a streamlined process with one fine-tuning is used in Fast R-CNN method. In this novel method, similar to the previously introduced R-CNN method, category-independent regions or object suggestions are removed from the view. Then, the entire input image and a set of object proposals are taken by the Fast R-CNN network. The entire image is first processed by the network with several convolutional and max pooling layers to produce a convolution feature map. Then, a fixed-length feature vector is extracted by a RoI pooling layer from the feature map for each object proposal and each feature vector is then fed into a sequence of fully connected layers. Python and C++ are used to develop the Fast R-CNN. Three ImageNet models are used in the experiments: AlexNet model, VGG_CNN_M_1024, which has the same depth as AlexNet model but is wider and Very deep VGG16 model. The proposed approach is put to the test on three different datasets: VOC 2007, VOC 2010 and VOC 2012. The top result is achieved on VOC 2012 with a mAP of 65.7 percent. In the very deep VGG16 network, Fast R-CNN is trained 9 times faster than R-CNN, 213 times faster at test time, and had a higher mAP in PASCAL VOC 2012. Also, compared to SPPnet, Fast R-CNN also trains VGG16 3 times faster, tests 10 times faster, and is more accurate than SPPnet. Due to the significant

improvement in object detection time with the Fast R-CNN, many studies are carried out using this method. In previous studies in the literature, the Fast R-CNN method is used to detect traffic signs. As a good example of these studies, Boujemaa et al. [23] worked on a TSR system which utilized the Fast R-CNN approach. The study is focused on how to detect and recognize traffic signs while driving. As a result of these features, the system can help drivers avoid danger by guiding and warning them. Two different methods are used in this study: Color segmentation-CNN (C-CNN) and Fast R-CNN methods. The C-CNN technique involves reducing the search space by applying a color threshold to the input image and identifying a set of regions of interest (RoI). After that, the RoI is classified using a trained CNN, following that, a similar CNN is used to recognize the traffic signs that are detected. To enhance training and testing speed as well as detection accuracy, the Fast R-CNN approach employs a number of strategies. German Traffic Sign datasets are utilized to evaluate the proposed models and compare the performance of the implemented methods, for detection (GTSDB) and for recognition (GTSRB). Python language is used for implementation and the OpenCv API is used for color segmentation. The well-known deep learning API Keras with Tensorflow backend are used to train the CNNs models in both approaches. The C-CNN approach results showed CNN1 and CNN2 accuracies were 93 percent and 95 percent respectively. The Fast R-CNN approach results showed 94.8 percent accuracy on the test set (GTSDB). In terms of speed, the Fast R-CNN is proved to be far superior to the C-CNN. It is also unaffected by changes in lighting. Another study to detect traffic signs is conducted by Qian et al. [24]. Since the previous studies on this subject were mostly aimed at detecting the signs around the road, this study focuses on the detecting of the signs on the surface of the road. This proposed study is divided into two parts. With color and edge information, a hybrid region proposal method is utilized to predict the location of traffic signs in the first part. The Fast R-CNN approach is utilized to perform classification and bounding box regression simultaneously in the second part. In this study, a dataset of field-captured road surface traffic signs is employed. It consisted of 2223 images including 4204 traffic signs. In the implementation part, the program is run on a 64-bit open source Linux operating system with CUDA 7.5. Python is used as a programming language and the Fast R-CNN models are trained on the Caffe deep learning platform. In this study, two experiments are set up for testing the performance of multitask training and bounding box regression. The results show that the overall average precision (AP) is about 85.58 percent for the Fast R-CNN with bounding box and 83.99

percent for the Fast R-CNN without bounding box. Although the proposed study has achieved high performance, there are still undetectable signs during the test due to scale variations, and colors fading. It could be solved with a more accurate region proposal algorithm.

2.3.3. Faster R-CNN Approaches

In the processes mentioned in the previous section, a selective search method is used to generate region proposals. Since the selective search method is slow and time consuming, it negatively affects the performance of the networks. Ren et al. [25] noticed that the convolutional feature maps used by region-based detectors such as Fast R-CNN can also be used to generate region proposals. Thus, a novel object detection algorithm Faster R-CNN is introduced. In this algorithm, instead of using selective search method, a new Region Proposal Network (RPN) is used to share full-image convolutional features with the detection network. RPN and Fast R-CNN networks are merged into a single network and trained to learn the region proposals due to providing less costly region proposals. The Faster R-CNN is a single, combined network including two modules for object detection. The first module is RPN which proposes regions and the other one is Fast R-CNN detector using RPN's proposed regions. In the training process, the PASCAL VOC dataset is used and Caffe is used for implementation. The experiments are performed on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image and it showed that RPNs with Fast R-CNN generated detection accuracy better than the Fast R-CNNs which use selective search. The results are achieved on VOC 2007 with a mAP of 78.8 percent, on VOC 2012 with a mAP 75.9 percent and on MS COCO datasets, Faster R-CNN performed 2.2 percent higher than Fast R-CNN with 42.1 percent accuracy. Furthermore, Faster R-CNN gave higher accuracy, it is more than 10 times faster than Fast R-CNN at test-time. Moreover, several first places are achieved by using Faster R-CNN in ILSVRC & COCO 2015 competitions. After the introduction of the Faster R-CNN, the name of this method has started to be mentioned frequently in object detection and recognition studies. Better results are obtained with this method in challenging problems in recognizing traffic signs. For example, detecting small traffic signs is a challenging problem and it is also very important to detect accurately to ensure safety in smart vehicles. For this, Zhang et al. [26] proposed an improved RPN architecture to apply the Faster R-CNN on the challenging

task of small traffic sign detection since applying Faster R-CNN directly provides an unsatisfactory detection rate in detecting small objects. In this study, a receptive field guided RPN is introduced to boost proposal quality and in the R-CNN detection subnetwork, a fully convolutional network is used instead of a fully connected network to reduce model size and keep the accuracy. CCF BDCI 2016 Traffic sign detection in self-driving scenarios is used as a dataset which contains 4000 images and about 28000 traffic signs. For training 3000 randomly selected pictures were used and the rest were for validation. In this study proposed Receptive Field net (RFnet), Faster R-CNN and SSD were compared. The results showed that RFnet performed the best, achieving 50 percent AP value and it had a 5.4 percent higher accuracy rate than Faster R-CNN. Also, the SSD achieved 41.9 percent AP accuracy, which is smaller but closer to the performance of the Faster R-CNN with 44.6 percent AP. In terms of detection speed, the fastest method is Faster R-CNN with 0.14 s and the proposed RFnet is able to achieve a test speed of 0.39 s. Although the proposed method is higher in accuracy, it is slower than the Faster R-CNN method in terms of speed. Another study on this challenging topic which is detecting small traffic signs is conducted by Yang et al [27]. A novel framework is proposed for real-life traffic sign detection. The proposed framework combined traditional computer vision technology with Faster R-CNN to adapt to detect small targets in large and complex backgrounds. A new Attention Network (AN) module integrated to Faster R-CNN to constrain the searching area and tell the system where to look according to the colour characteristics of the traffic signs. In this study, AN is proposed which extracts region proposals in a coarse-to-fine manner to avoid missing small targets. Thus, the powerful feature extraction capability of the CNN is integrated with the use of all the characteristics of the target. After extracting the region proposals, a detector improved from Fast R-CNN is used to perform the final classification and regression task. In the experiments, the proposed AN module largely reduced the search range to reduce the time and improved the accuracy of detection results. The proposed method is evaluated over two benchmarks: Tsinghua–Tencent 100K Traffic Sign Benchmark (TT100K) for Chinese traffic signs and Belgium Traffic Signs Dataset (BTSD) for Belgian traffic signs. These datasets contained 9180 and 9007 images, respectively. During the test phase, traffic signs to be detected are divided into three categories according to size in terms of pixels: small targets whose areas are less than 32x32 , medium targets whose areas are between 32x32 and 96x96 and large targets whose areas are more than 96x96. The test results showed that the proposed method

performed well in terms of both speed and mAP. The proposed method performed better on small sized object targets, which reaches the an AP of 49.81 percent, 18.59 percent higher than Faster R-CNN for small targets in TT100K, and 50.82 percent, 6.89 percent higher than Faster R-CNN in BTSD. Also, in terms of speed, the proposed method with ZFnet took in total 128ms, 37ms faster than Faster R-CNN. While with VGG-16, it took 458ms, 77ms faster than the Faster R-CNN.

3. METHODOLOGY

This chapter highlights region-based CNN architectures. This chapter covers the fundamentals of deep learning approaches for object recognition and image classification. First, the basics of computer vision is discussed in section 3.1, followed by ANNs in section 3.2 and section 3.3 discusses the CNNs. Finally, in sections 3.4, 3.5, and 3.6, the essentials of region-based CNN are discussed.

3.1. COMPUTER VISION

Computer vision (CV) is one of the most popular topics nowadays and the number of studies in this field is increasing everyday. The fact that there are many rapidly growing beneficial real world applications based on this field of study increases the importance of CV. We are experiencing these applications during the day without realizing it. CV technology is based on mimicking the way the human brain works. DiCarlo et al. [28] conducted a study on how the human brain recognizes objects and in this study, it was proposed that the brain uses patterns to recognize objects. CV, like the human brain, uses patterns to recognize objects. By using patterns, meaningful information can be extracted and interpreted from videos and photos through algorithms. To do this, it is necessary to train a computer to recognize visual data by feeding it with a fairly large number of labeled digital images. Then, the algorithms allow the computer to find patterns in all digital images related to these labels. A digital image consists of numbers at a specific location on a grid of pixels which are called bitmaps. As seen in Figure 3.1, grayscale pictures consist of a single channel. Grayscale images can be represented in 2-dimensional matrices, and each value of the matrix consists of a single 8-bit number which represents the brightness of each pixel. On the other hand, color pictures consist of 3 channels. These channels are red, green and blue. Since color images contain 3 channels, unlike grayscale images, they can be represented in 3-dimensional matrices.

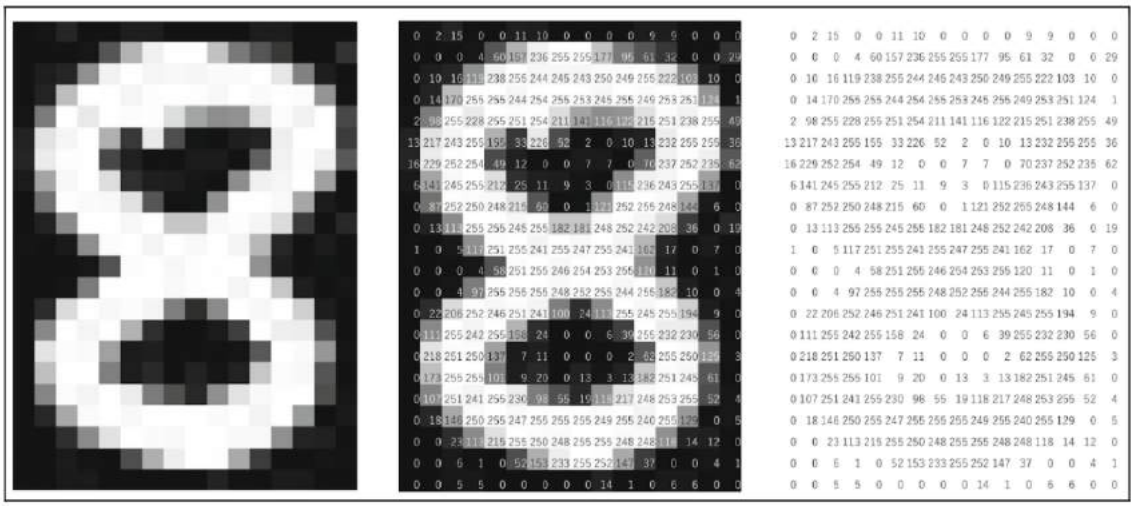


Figure 3.1. Example of two-dimensional grid [29]

For extracting and interpreting meaningful data from the digital image, pixel-level operations are performed. Pixel-level operations are the foundation of many image processing algorithms. Converting an image to grayscale or scaling an image are examples of pixel-level operations. There are many significant CV techniques that can assist a computer extract meaningful information from images. Object detection, for example, recognizes a specific object in an image. Another example is image classification, which classifies images into categories. These techniques are frequently used in detecting and recognizing traffic signs, especially in advanced driver-assistance systems. The use of CV is critical in fulfilling the goals of these systems.

3.1.1. Object Detection

In order to interact with an environment, it is necessary to recognize objects by detecting them. The process of detecting and recognizing different objects for the human brain may seem naturally effortless due to years of learning. However, this process is quite challenging for computers because of the problem of localization and classification of objects in visual data. Object detection, which is the process of detecting and localizing various objects within visual data, is one of the most essential CV approaches. To do this, it predicts the location of a certain class of objects within the input visual data. Bounding rectangular boxes are also drawn around the objects to help find where the predicted

objects are within the visual data. For example, using the object detection technique, information about where and how many objects belonging to a certain class can be obtained in visual data as seen in Figure 3.2.

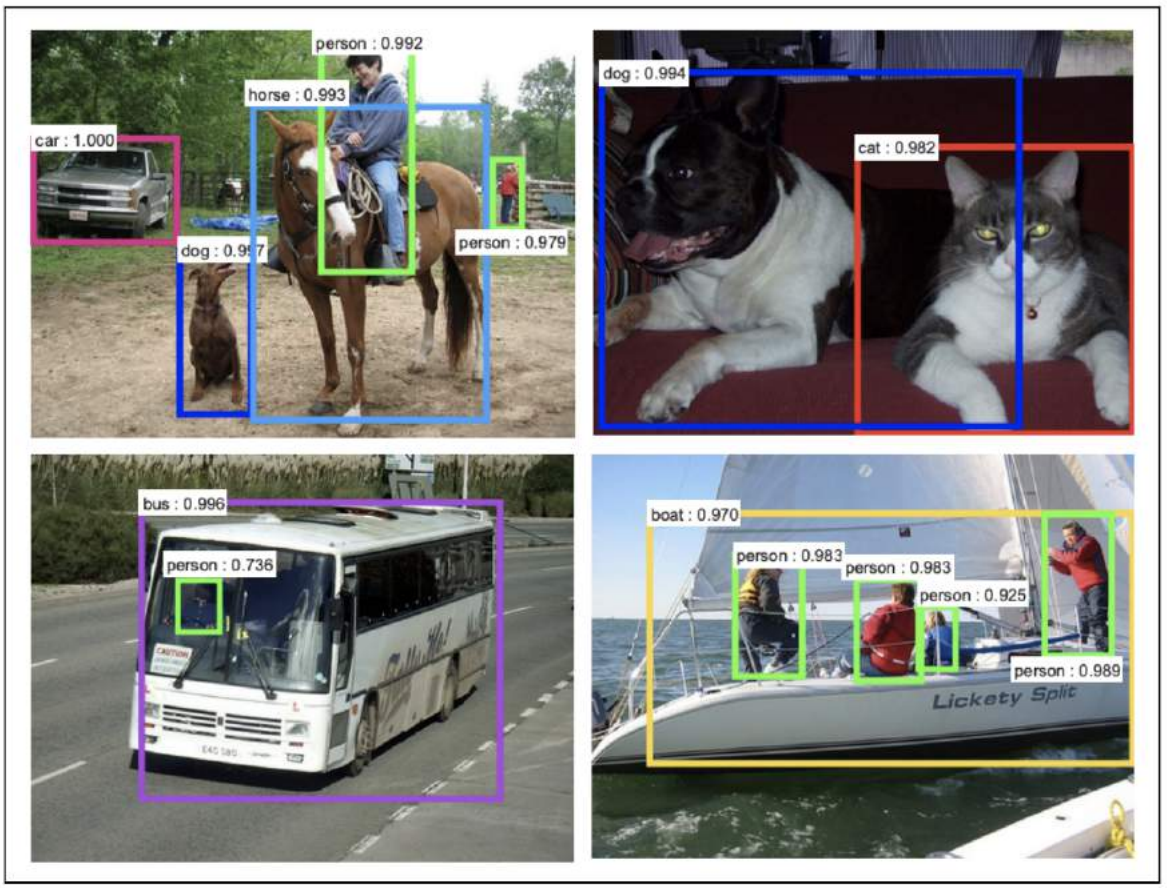


Figure 3.2. Example object detections on PASCAL VOC 2007 test [25]

In the literature, several techniques have been proposed via various studies conducted on object detection throughout the years. These techniques can be split into two categories. The first one is non-neural network based techniques and the other is neural network based techniques. In non-neural network based methods, the object detection technique recognizes the pixel group of the object by looking at the features of the visual data. Then, it estimates the object's position along with its label with these features. On the other hand, in neural network based methods, object detection is performed using convolutional neural networks. In this method, there is no need to define features. Neural network based techniques have become cutting-edge approaches to object detection.

3.1.2. Image Classification

When people look at visual data, they can usually recognize what it represents without difficulty. This is because the information one has learned from an early age makes it easier for a person to recognize what the visual data depicts. However, when visual data is requested to be recognized by computers, this will not be as easy as it seems. Because of the fact that this process does not work on computers like that of humans. When it comes to image classification, it is still challenging for computers. One of the most significant CV techniques is image classification. It's a task that necessitates the use of an algorithm to figure out which object classes are present in the image. [11]. Usually image classification models are trained to take visual data as input and return a label based on the content of the visual data as a result. Also, image classification models return a confidence score of which class the visual data belongs to, as well as predicting the content of the visual data. In Figure 3.3, a cat picture was given as an input to the image classification model. As a consequence, the cat tag is returned with a score of 82 percent.

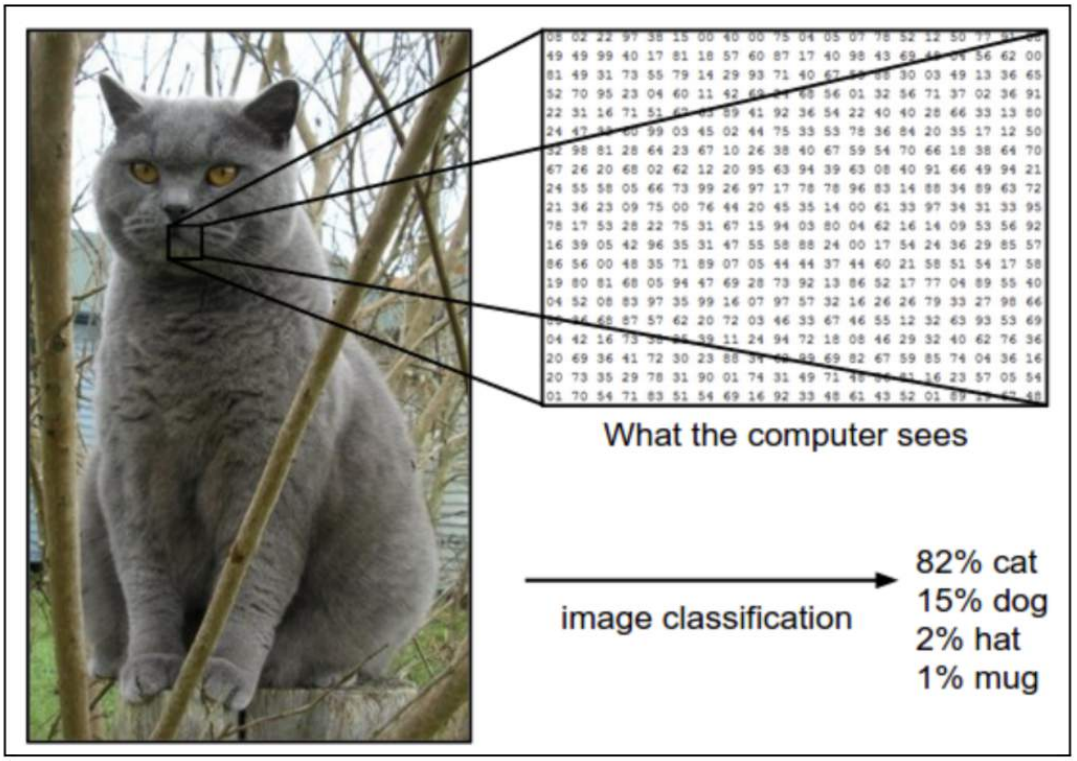


Figure 3.3. Example image classification [30]

3.2. ARTIFICIAL NEURAL NETWORK (ANN)

Over many years of study, the human brain's complexity has remained unsolved. The way the human brain interprets the situation is still a challenging problem for computers to address. To solve this problem, the functioning of the human brain has been tried to be imitated in computers. To do this, it is inspired by biological neural networks. ANN is a complex network that works similar to a human brain [31]. ANNs imitate the network of neurons that make up the human brain so that computers can learn and make decisions like humans. Learning mathematical functions through an iterative process is the purpose of an ANN. In 1943, McCulloch and Pitts [32], demonstrated a similarity between biological neurons and simple logic gates with binary outputs. Neural networks in a biological brain are formed by the combination of neurons. Neurons are the components of an ANN that execute computational operations.

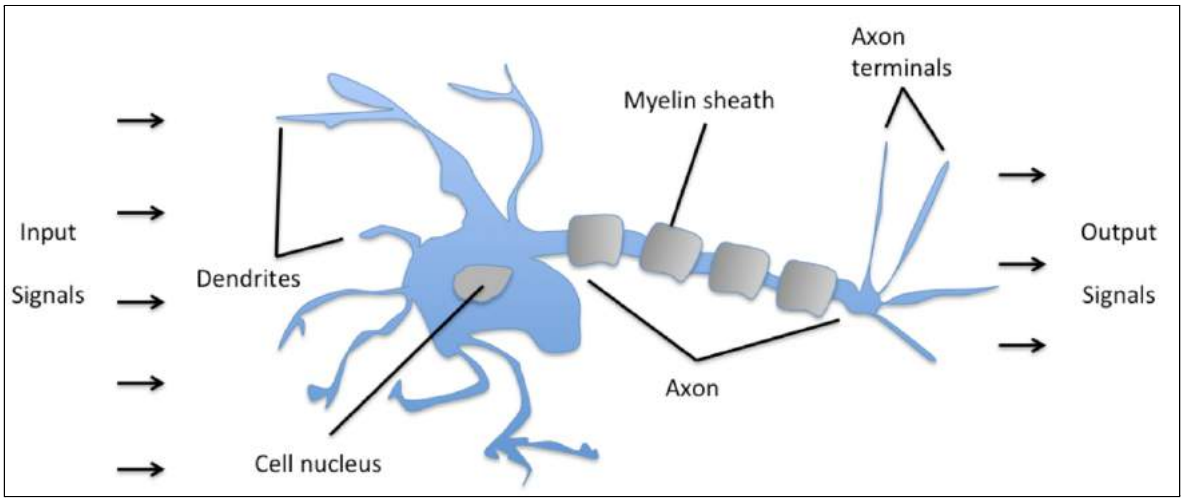


Figure 3.4. Schematic of biological neuron [33]

As seen in the Figure 3.4, signals of variable magnitude reach the dendrites, and the signals coming through the dendrites accumulate in the cell body of neurons. When the sum of these signals reaches a particular threshold, an output signal is produced that passes through the axons. In this way, neurons can transmit a signal to other neurons. A neuron that receives a signal can process it and send a signal to the neurons connected to it. In a study conducted in 1958, Rosenblatt [34] introduced the single layer neural network called perceptron, which is the first modern neural network. The fundamental unit of a neural

network is the artificial neuron, commonly known as a perceptron. Rosenblatt' study showed that perceptrons can actually learn from data. In this proposed neuron model, the artificial neuron was enabled to figure out the correct weights directly from training data by itself. The perceptron, known as the smallest part of the artificial neural network introduced by Rosenblatt, is a mathematical modeling of a human nerve cell.

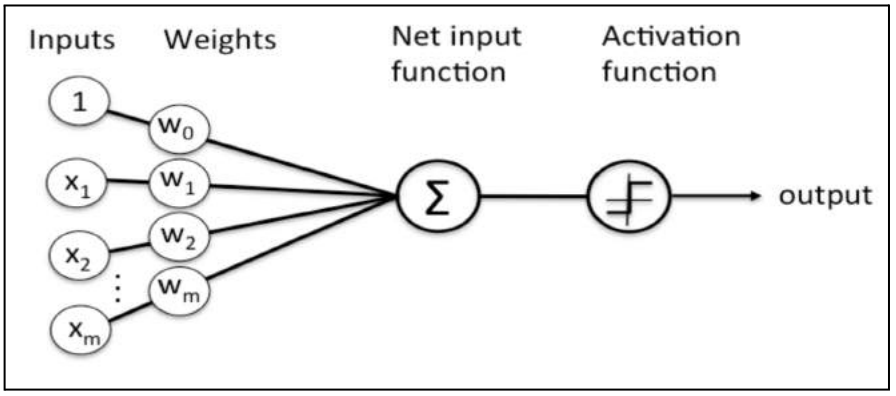


Figure 3.5. Schematic of Rosenblatt's perceptron [34]

As seen in the Figure 3.5 perceptron is quite similar to the one proposed in 1943 by McCulloch & Pitts. Perceptron consists of 4 parts. In the inputs part, input values are transmitted to a neuron. It resembles a biological neuron's dendrite. In the weights part, input weights are multiplied to the respective input values. The sum of all these multiplied values is called the weighted sum. Then, the bias value is added to the weighted sum value to get the final value for prediction by the neuron. In the activation function part, it is decided whether a neuron can be activated or not. Finally, in the output part, the final output of a neuron is given, which can be passed to other neurons in the network or taken as the final output value. In the following years, multi-layer neural networks were used instead of single layers. Thus, deep learning has emerged. Deep learning is based on the concept of numerous processing layers. Ivakhnenko and Lapa [35] created a deep learning model in 1965, as illustrated in Figure 3.6.

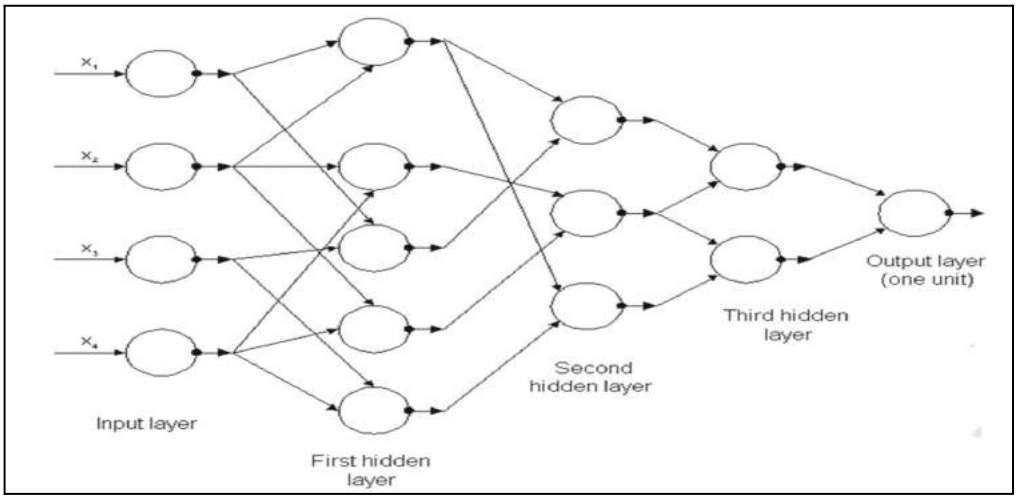


Figure 3.6. The architecture of the first known deep network [35]

A deep neural network with three hidden layers is shown in Figure 3.6. The input layer's values are used by the first hidden layer to make decisions. Then, the second hidden layer will make decisions depending on the first hidden layer's outputs. Like the second hidden layer, the third hidden layer eventually makes decisions depending on the prior layer. In ANN, as the number of hidden layers increases, the number of parameters included in the calculation also increases. Calculating a large number of parameters in a network is quite complex with a classical ANN model. Thus, the inadequacy of classical ANN models in calculating many parameters brought different perspectives. As a solution to this inadequacy, CNN with LeNet architecture was used in 1998 by Yann LeCun et al. [10]. In section 3.3, CNNs are discussed in detail.

3.3. CONVOLUTIONAL NEURAL NETWORK (CNN)

Although the first studies on deep learning started many years ago, one of the main reasons for its successful use in recent years is the enormous increase in the number of data. Although deep learning algorithms, which are used in complex tasks today, are almost the same as classical artificial neural network algorithms, in the models prepared with deep learning algorithms, very deep architectural changes have been made in order to simplify the training. Thus, it became a solution to the computational complexity brought about by the increasing number of data today. CNN with LeNet-5 architecture was used in 1998 by Yann LeCun et al. and this network is considered to be the first successful implementation

of convolutional networks. A CNN is a feed-forward multilayer perceptron. CNN's main idea was inspired by a feature of the brain's visual cortex called receptive field [36]. CNNs' biggest advantageous feature is that they reduce the number of parameters in ANN. CNN is the one among the most popular deep learning approaches, which achieved very significant results in some complex visual tasks. CNNs consist of multiple layers placed one after the other for feature extraction. The layers identify lower-level features such as curves and edges and build them up to more abstract concepts as seen in Figure 3.7. The continuation of these layers is a trainable classifier.

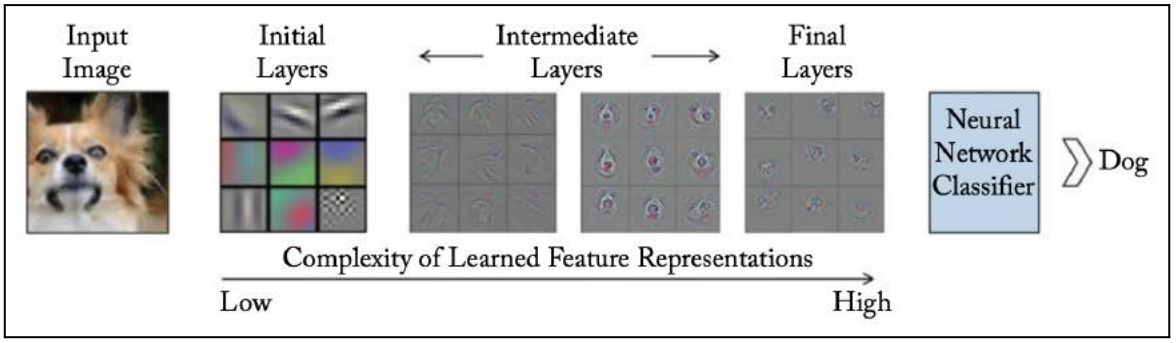


Figure 3.7. Different representations of the object on different layers in the network [37]

In CNN, after the input data is received, the layer-by-layer training procedure is performed. Finally, it outputs a final result that may be compared to the correct result and then, an error happens equal to the difference between the generated result and the expected result. By updating the weights at every iteration, the error is reduced.

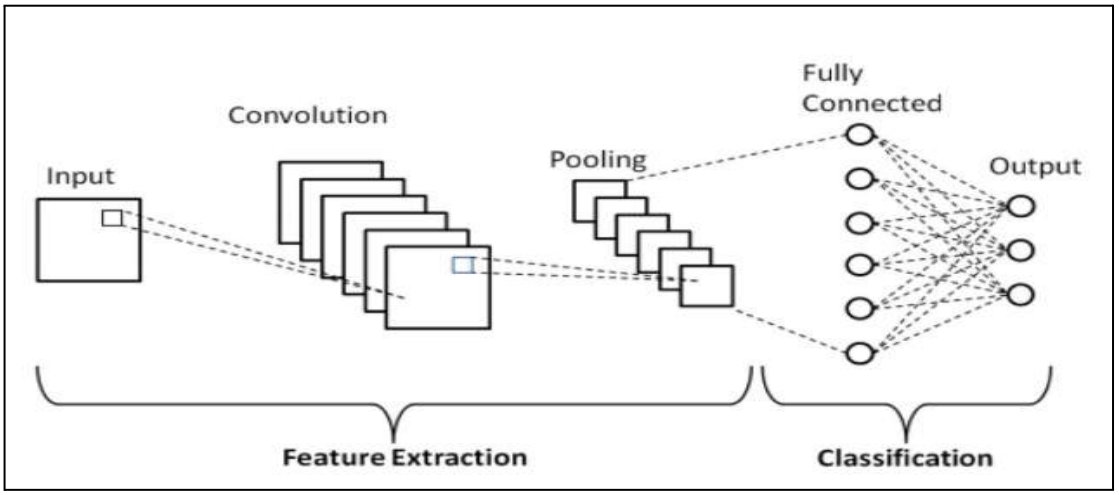


Figure 3.8. A basic diagram of a CNN architecture [38]

The basic architecture of the concept of deep learning is considered to be CNN architecture. A typical CNN is constructed by input layer, output layer and the repetition of three basic types of layers, namely convolutional layers, pooling layers, and fully connected layers as seen in Figure 3.8. Each layer has its own function. Detailed information about the functions of the layers of CNN is given in the subtitles of this section.

3.3.1. Layers of CNN Model

The input layer is the first layer of the whole CNN. In this layer, data is given raw to the network. The input layer in CNN should contain image data, and the image data is usually represented by a matrix that holds the pixel values of the image. It also has a depth that represents color channels. Generally, three for RGB color channels. Choosing a high size input image to be fed into the network can increase the memory requirement on the network and cause longer training time. However, due to the increase in the depth of a network fed with a high size input image, its performance may increase. In CNNs, an appropriate input image size should be selected for both hardware computing cost and network success.

A convolutional layer is the fundamental component of a CNN. In the convolutional layer, the input to the convolution layer is convolved with what is called a kernel, a convolution matrix, or a filter to generate an output feature map. Convolution process is based on the process of sliding a filter over the input image as seen in Figure 3.9. As a result of this convolution process, a feature map is generated. Feature maps are 2-dimensional matrices where specific features are discovered for each filter.

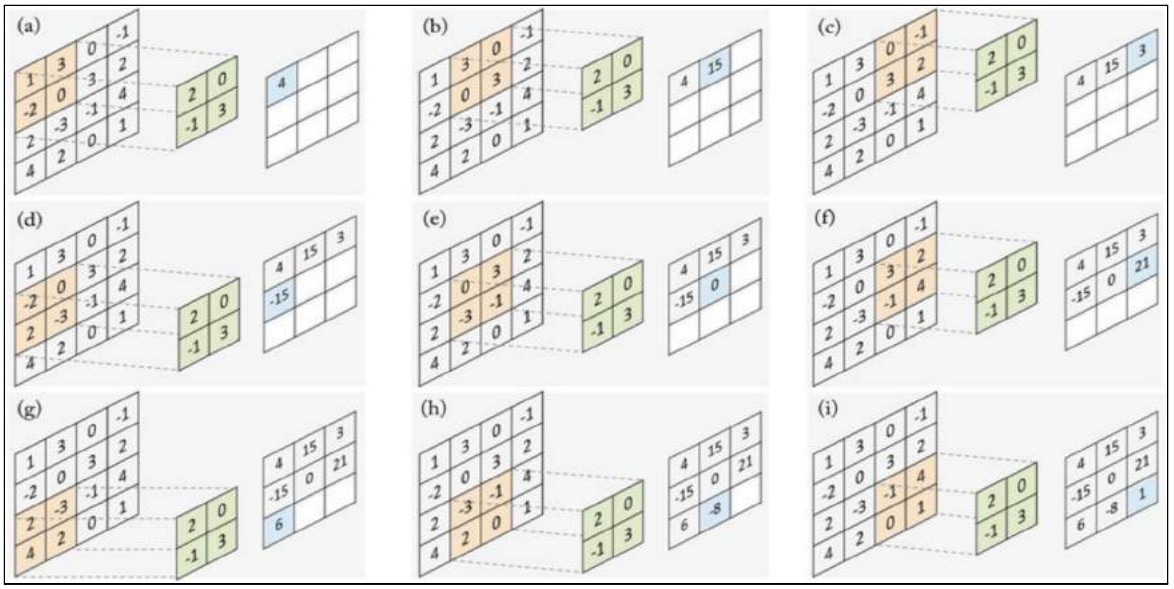


Figure 3.9. A convolution process with a 2x2 filter applied to a 4x4 input image [37]

As shown in Figure 3.9, the convolution process is implemented by sliding a filter on the input image. Images (a) through (i) show the calculations performed at each step. For the convolution process, a 2x2 filter (shown in green) is slided a certain step to the right or left on the input image. This step is termed as the stride of the convolution filter and its value is set to one in this example. First, the filter is positioned in the upper left corner of the image. Here, the indices between the same sized region (shown in orange) within 4x4 picture matrices and filter matrices are multiplied by each other and all results are summed, then the result is stored in the output matrix (shown in blue). Then, this filter is moved to the right by the stride value and the process is repeated. After the end of the 1st row, the 2nd row is passed and these calculations are repeated. After all calculations are finished, the output matrix is generated which is called a feature map. Generally, more than one filter is used to detect multiple features. For this reason, there are more than one convolutional layers in CNN.

Pooling layer is a layer often added between successive convolutional layers in the CNN. A pooling layer operates on blocks of the input feature map and combines the feature activations. The task of this layer is to reduce the spatial size of the data representation and the parameters and number of calculations within the network. Thus, it also controls overfitting. The operations taking place in the pooling layer do not affect the size of the depth in the data. Like the convolution process performed in the first step, the process in

the pooling layer is also based on the process of sliding a filter over the input image. The filter is slided a certain step on the input image and the feature activations are combined by taking the maximum values or average values of pixels in the image depending on the pooling operation. There are two commonly used pooling operations such as average pooling operation and maximum pooling operation. Maximum pooling operation is more preferred as it gives better results. In CNNs, the pooling layer is optional and many people do not prefer to use this layer. The larger stride is preferred on the convolutional layer instead of using the pooling layer.

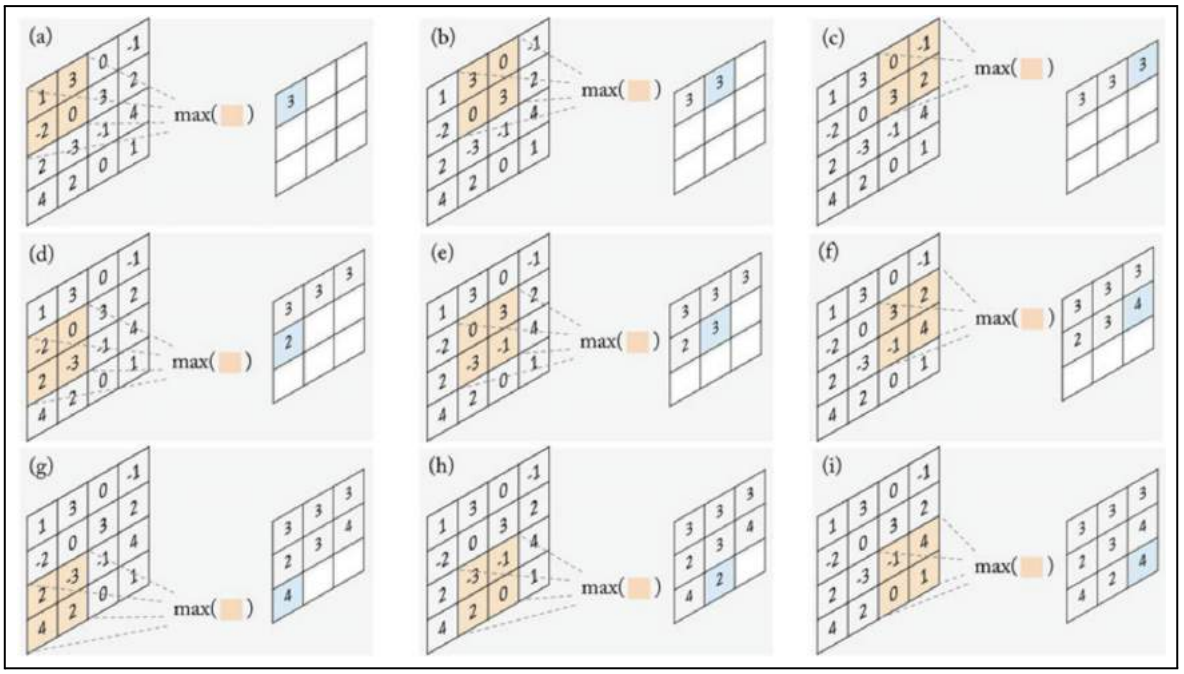


Figure 3.10. Max pooling operation with a 2x2 filter applied to a 4x4 input image [13]

As shown in Figure 3.10, The operation of max-pooling is implemented when the size of the pooling region is 2x2 and the stride value is 1. Images (a) through (i) shows the computations performed at each step as the pooled region in the input feature map (shown in orange) is slided at each step to compute the corresponding value in the output feature map (shown in blue). The max() pooling operation is taking the largest of four numbers in the filter area. In this way, it uses smaller outputs that contain enough information for the neural network to make the right decision.

Fully-connected layer is the last and most important layer of CNN that involves weights, biases, and neurons. The fully-connected layer is similar to the way that neurons are arranged in a traditional neural network. Each node in a fully-connected layer is directly connected to every node in both the previous and in the next layer as seen in Figure 3.6. In a typical CNN, a fully-connected layer is usually placed toward the end of the architecture and generally acts as a classifier. This layer is used to calculate the class scores to use as the output of the network. The number of this layer may vary in different architectures.

The output layer is the last fully-connected layer in CNN architecture and it represents the class scores. After multiple layers of convolution and pooling, the output is needed in the form of a class. Since the convolution and pooling layers would only be able to extract features and reduce the number of parameters from the input image, a fully connected layer is required to generate an output equal to the total number of classes in order to generate the final output.

3.3.2. Popular Architectures of CNN Model

LeNet-5 is a deep learning model based on CNNs that was first published in 1998. It was developed by LeCun et al [10] for reading numbers on postal numbers, bank checks, simple digits, etc. and experiments were carried out on the MNIST dataset [39]. In this model, unlike other models that will be developed later, average pooling is performed instead of max-pooling in the pooling layer. The LeNet-5 model consists of two convolution layers each followed by a sub-sampling (max-pooling) layer to extract features. Afterward, a single convolution layer, followed by a set of two fully connected layers toward the end of the model to act as a classifier on the extracted features as seen in Figure 3.11.

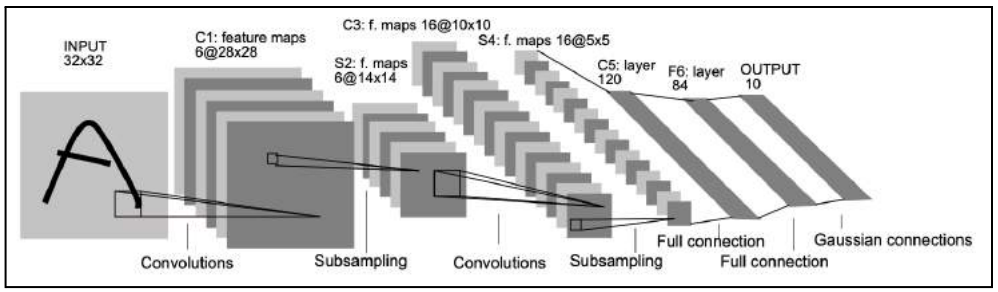


Figure 3.11. The architecture of the LeNet-5 CNN model

AlexNet made a major contribution to popularizing CNN models and deep learning again. It is the first large-scale CNN model introduced by Alex Krizhevsky et al. [12] is similar in its architecture to Yann LeCun's LeNet-5 model due to the presence of successive convolutional and pooling layers as seen in Figure 3.12. Unlike the LeNet-5 model, in this model, max-pooling is used in pooling layers. This architecture won the ILSVRC [11] in 2012 by a large margin. The CNN model is renamed as AlexNet after its performance at the competition. It is a breakthrough in the image classification problem by providing a sudden increase in classification accuracy from 74.3 percent to 83.6 percent in ImageNet ILSVRC competition.

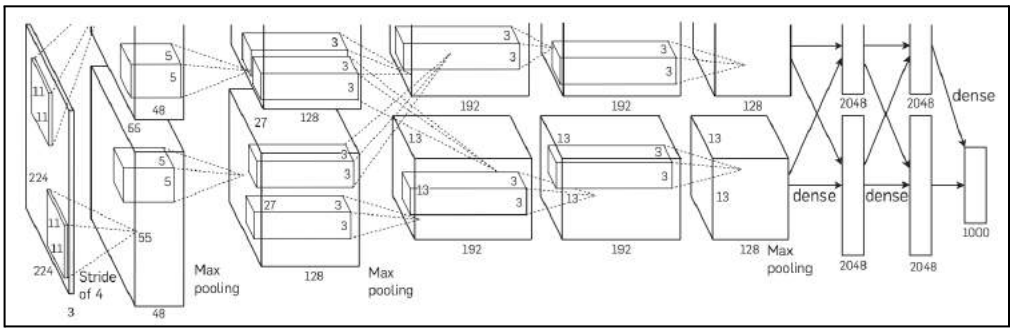


Figure 3.12. The architecture of the AlexNet CNN model

VGG-16 is first proposed by Simonyan et al. [40] and was runner up in the 2014 ImageNet Challenge, showing that small convolutional filters and increased network depth can significantly improve performance. It is a simple network model and the most important difference from the previous models is the consecutive use of 2 or 3 convolutional layers as seen in Figure 3.13. The input to the first convolutional layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers. Three fully connected layers follow a stack of convolutional layers. The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. ReLU activation function is used in this network. In total the VGG-16 architecture has around 138M parameters. In the ILSVRC, the VGG-16 architecture achieves the best test error result of 7 percent. However, there are two major drawbacks with the VGG-16 model. The first one, training is extremely slow and the second one, it is more costly to evaluate and uses a lot more memory.

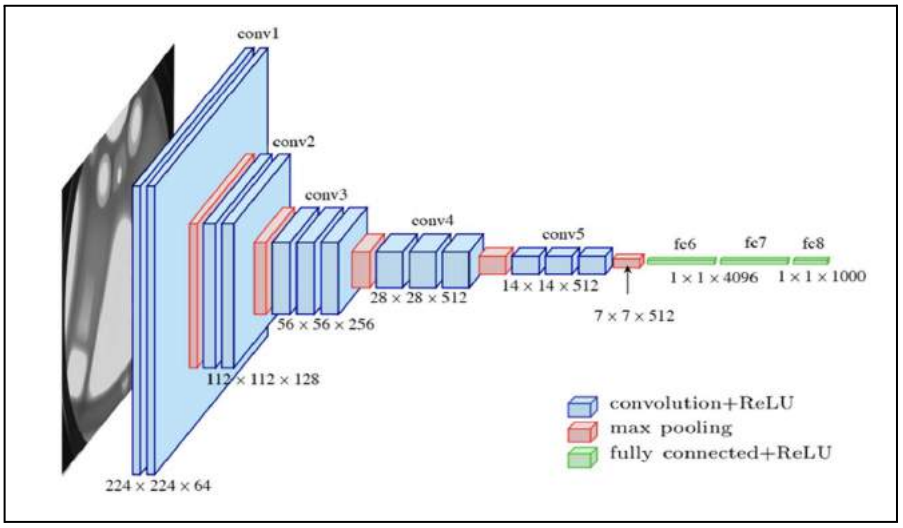


Figure 3.13. The architecture of the VGG-16 model

MobileNets are introduced by Google [41][42] in 2017 and they are neural network architectures that perform very well on mobile devices. MobileNets are commonly used for image classification. They are models based on CNN. The MobileNet architecture has the advantage of requiring less computation than the traditional CNN model. To develop lighter models, MobileNets generally employ depthwise separable convolutions rather than the standard convolutions utilized in previous architectures. In MobileNet V2, a few key adjustments are made to the MobileNet architecture, which resulted in a large boost in model accuracy. The key architectural changes are the addition of inverted residual blocks and linear bottlenecks, as well as the adoption of the ReLU6 activation function instead of ReLU. The comparison of convolutional blocks for MobileNets as seen in Figure 3.14.

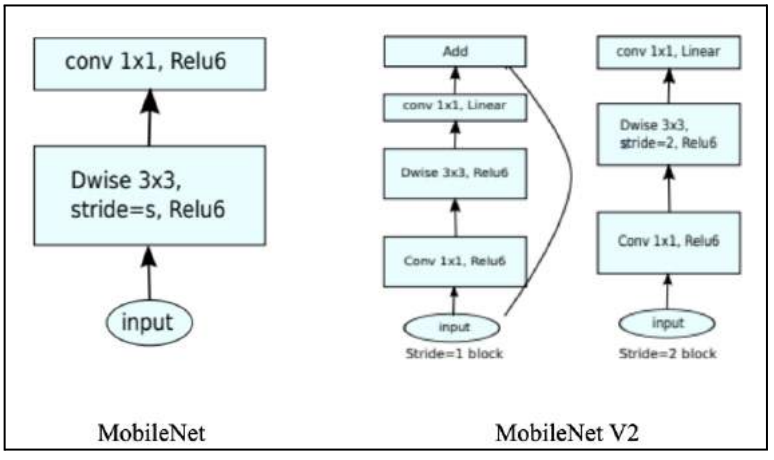


Figure 3.14. Comparison of convolutional blocks for MobileNets [42]

3.4. REGION-BASED CONVOLUTIONAL NEURAL NETWORK (R-CNN)

The R-CNNs were first developed by Girshick [20], and it is a ground-breaking approach to object detection that uses deep models. R-CNN architecture has been developed since the CNN algorithm is not sufficient to detect objects for visuals containing objects in more than one class. The R-CNN architecture is used to identify the classes of objects in the visuals and the bounding boxes belonging to these objects. The main idea in this architecture consists of two steps. In the first step, it identifies the object region candidates using selective search in the visual. In the second step, it extracts CNN features from each region separately for classification. The R-CNN model consists of three modules as seen in Figure 3.15.

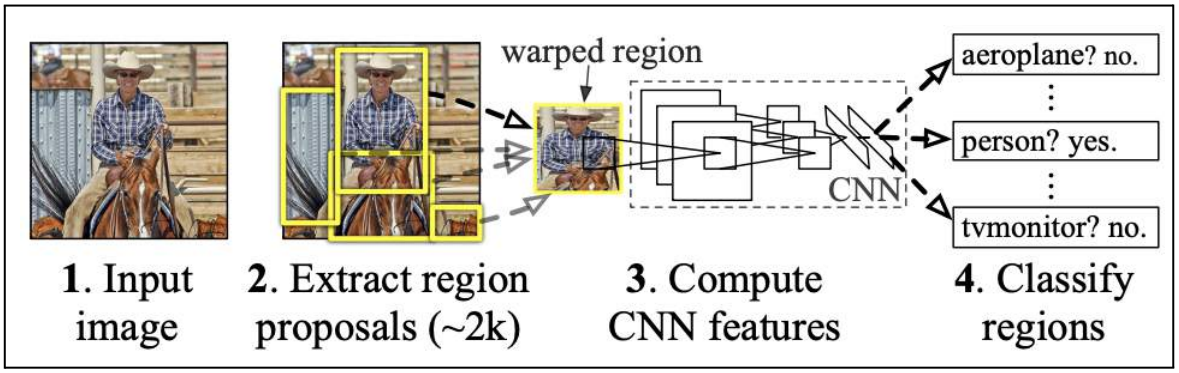


Figure 3.15. The architecture of the R-CNN model [20]

As shown in Figure 3.15, in the first module an image is taken as input then region proposals are extracted using selective search on this image. Approximately 2k candidate regions are proposed per image. Since these regions include target objects of varying sizes, region candidates are warped to have a fixed size as required by CNN. In the second module, a massive CNN extracts a fixed-length feature vector from each region. The third module is a set of class specific linear SVMs. In this module, the features and labeled category of each proposed region are combined and if there was an overlap in regions, the proposal with the higher score calculated by the SVM has been taken. This step was done independently for each object class. Although the R-CNN was ingenious, it had few drawbacks. One of the significant drawbacks was the region proposals' fixed input size which was overcome by warping image regions to predefined dimensions before processing by the CNN. Another major drawback was the high computational cost of

R-CNN due to the generation of approximately 2k regions at test time. Since CNN is applied to each region proposal in the image separately, the training time takes approximately 84 hours and the estimation time approximately 47 seconds. Because of the high time cost, this model is unlikely to be used, particularly in real-time object detection applications.

3.4.1. Selective Search Method

Selective search [43] is a method used to generate region proposals for detecting objects in visuals. This method works by computing hierarchical groupings of identical regions based on colour, texture, size, and shape similarity, and in this method, small regions are determined first. Then, two similar small regions are combined to generate a larger new region. At each iteration, larger regions are produced and added to the list of proposed regions. Thus, region proposals are generated from smaller segments to larger segments with a bottom-up approach. In the R-CNN model, certain region candidates in the visual are identified using selective search. Approximately 2000 different regions emerged at the end of this process. Each of these region candidates is fed into a different CNN network, and 2000 CNN networks are used for these 2000 regions. The class of the object is determined in SVM models by using features which are extracted from CNN networks.

3.4.2. Intersection over Union

The region of overlap between two boxes is described by the term intersection over union (IoU). The higher the overlap region, the higher the IoU. It is mostly utilized in object detection applications, where a model is trained to produce a box that properly fits around an object. In Figure 3.16, for example, there are two boxes: one green and one red. The green box represents the ground truth, while the red box represents the prediction. The purpose of the model is to continue improving its prediction until the red box and green box overlap perfectly, that is, until the IoU between the two boxes is equal to 1.

Figure 3.16. An example of ground-truth box and prediction box

The IoU is calculated by dividing the area obtained by the intersection of the boxes by the area formed by the union of the boxes. The formula, which is also used to calculate the IoU, is given in Figure 3.17.

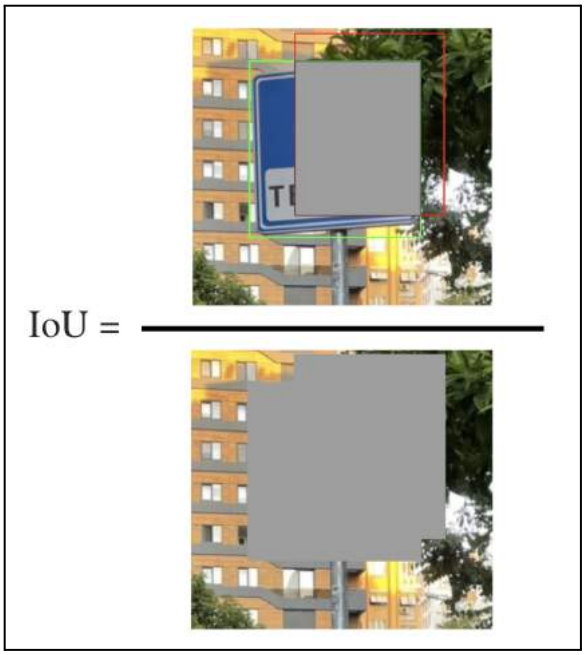


Figure 3.17. The formula of IoU

3.4.3. Non-Maximum Suppression

In object detection, the model can produce several predictions for an object. While each detection may be correct, it might lead to some problems if the classifier detects more than one item when it is obvious that there is only one. The Non-Maximum Suppression (NMS) approach, also known as Non-Maxima Suppression, is used to solve this issue. Non-Maximum Suppression is a technique that chooses one entity from a set of overlapping entities in the computer vision field. Estimates that are less than a specific level of probability are often eliminated from the criteria. The estimate with the highest probability is chosen from the other estimations and is considered a prediction. Other estimates lower than the threshold value are rejected. As a result, choosing a threshold value is critical to the model's success.

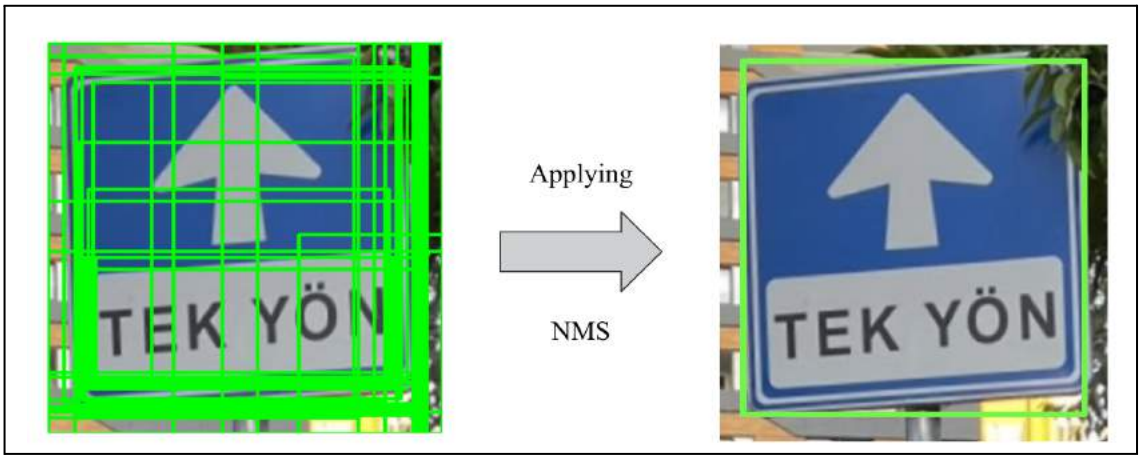


Figure 3.18. An example of the implementation of NMS

In Figure 3.18, more than one bounding box has been drawn on the B-16 (one-way) traffic sign. Although most of these predictions are valid, since there is only one traffic sign in the picture, the NMS technique is used to minimize the number of predictions to one. The most often used algorithm for this task is NMS.

3.4.4. Bounding Box Regression

Bounding Box Regression (BBR) is used to improve localization performance. After using a class-specific detection SVM to score each selective search proposal, a class-specific BBR was used to predict a new bounding box for detection. The first instance of BBR was used in the deformable part models (DPM) [44]. Later, it was adapted for R-CNN [20]. While these two approaches are similar, the main difference between them is that the regression relies on image features computed by CNN for the proposal. The aim of using BBR is to learn a transformation that maps a proposed box to a ground-truth box where P is the proposal, G is the ground truth, and $\hat{G}$ is the predicted bounding-box. Each box is specified as a set of top-left coordinates and a width and height in pixels, where P = (P_x, P_y, P_w, P_h). All the transformation functions, $d_x(P)$, $d_y(P)$, $d_w(P)$, and $d_h(P)$, take P as input and transform to predicted bounding-box $\hat{G}$ by applying the transformation.

$$\hat{G}_x = P_w d_x(P) + P_x \tag{3.1}$$

$$\hat{G}_y = P_h d_y(P) + P_y \tag{3.2}$$

$$\hat{G}_w = P_w \exp(d_w(P)) \tag{3.3}$$

$$\hat{G}_h = P_h \exp(d_h(P)). \tag{3.4}$$

As shown in equations 3.1 and 3.2, $d_x(P)$ and $d_y(P)$ specify a scale-invariant translation of the center of P's bounding box while, as shown in equations 3.3 and 3.4, $d_w(P)$ and $d_h(P)$ specify log-space translations of the width and height which produce better results on the wide range of bounding-box dimensions. Each function $d_\star(P)$, where $\star$ is one of x, y, h, w, is modelled as a linear function of the $pool_5$ features of proposal P, denoted by $\Phi_5(P)$. Thus, $d_\star(P) = w_\star^T \Phi_5(P)$ can be obtained, where $w_\star$ is a vector of learnable model parameters. $w_\star$ can be learned by optimizing the regularized least squares objective (ridge regression) as shown in equation 3.5.

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^{\mathrm{T}} \boldsymbol{\phi}_5(P^i))^2 + \lambda \left\| \hat{\mathbf{w}}_\star \right\|^2 . \tag{3.5}$$

The regression targets $t_\star$, where $\star$ is one of x, y, h, w, for the training pair (P, G) are defined as shown in equations 3.6, 3.7, 3.8 and 3.9.

$$t_x = (G_x - P_x)/P_w \tag{3.6}$$
$$t_y = (G_y - P_y)/P_h \tag{3.7}$$
$$t_w = \log(G_w/P_w) \tag{3.8}$$
$$t_h = \log(G_h/P_h). \tag{3.9}$$

When using bounding-box regression, there are two important issues. The first issue is that for good results, high regularization was needed, so the value of λ was set to 1000 by Girshick [20]. The second issue is choosing which training pairs (P, G) to use. The task of transforming P to a ground-truth box G makes no sense if P is far from all ground-truth boxes. In R-CNN, only a predicted box with a nearby ground truth box with at least 0.6 Intersection over Union (IoU) is kept for training the BBR model.

3.5. FAST R-CNN

The generation of 2k different candidate regions in the R-CNN model, and the use of 2k different CNN networks for these 2k candidate regions cause a huge cost in terms of the training process. To solve the drawbacks of the R-CNN model, Girshick [21] introduced the Fast R-CNN in 2015. This approach is similar to the R-CNN algorithm but Fast R-CNN uses a main CNN with multiple convolutional layers to detect features in the picture before proposing regions. In this model, the input image is given to CNN to generate a convolutional feature map. Thus, there is no need to use a separate CNN for each region proposal. In Fast R-CNN, as in R-CNN, selective search is used to generate region proposals from the input image. The region proposals in the input image are mapped on the feature map obtained from CNN. Feature maps extracted from the input image should be the same size as they will be fed into a fully connected layer. Since the input images are of varying sizes, the RoI pooling layer is used to ensure that each feature map has the same fix-length representation. RoI pooling layer performs max pooling on each window of the feature map and produces a small feature map of fixed size. Then, the output of the RoI pooling layer passed through the two fully-connected layers to produce a RoI function vector. Finally, the network branches into two layers, a softmax layer predicts the class of the proposed region and another layer that predicts the four coordinates of the bounding boxes for all classes. Figure 3.19 illustrates the architecture of the Fast R-CNN model.

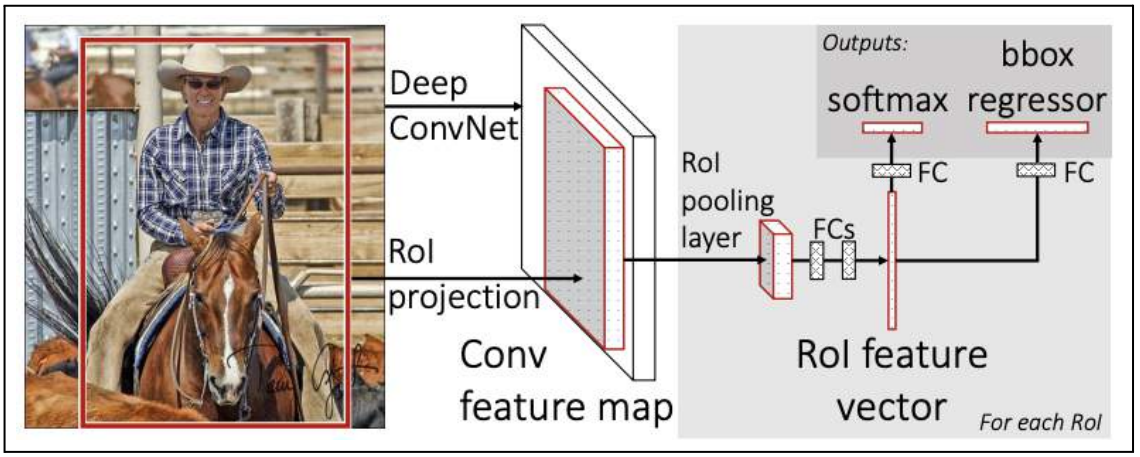


Figure 3.19. The architecture of the Fast R-CNN model [22]

Fast R-CNN is also faster in terms of both training and testing time. Because it is not required to feed 2k region proposals to the CNN every time. Instead, a feature map is generated from the convolution operation, which is done only once per image. For this reason, the training time takes approximately 8.75 hours and the estimation time approximately 2.3 seconds. It is 9 times faster at training and 213 times faster at test time compared to R-CNN. However, considering the large data sets in real life, this model is considered to be slow.

3.5.1. Region of Interest Pooling

The Region of Interest (RoI) Pooling layer is a crucial component of CNNs for object detection. During the proposal process, a large number of regions of interest are usually generated. It is because there is no way to correctly classify an object if it was not detected during the region proposal process. However, generating very large numbers of RoI could cause performance issues. This would make it impossible to apply real-time object detection. As a solution to this problem, RoI pooling was proposed by Girshick [22] and it significantly accelerates both training and testing. In the RoI pooling layer, max pooling is used to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of H × W. RoI pooling works by dividing the h × w RoI window into an H × W grid of sub-windows of approximate size h/H × w/W and then max-pooling the values in each sub-window into the corresponding output grid cell. Then apply max pooling in each feature map grid.

3.6. FASTER R-CNN

Selective search method is used to find region suggestions on R-CNN and Fast R-CNN models. This method is a slow and time consuming process that affects the performance of the network. In order to solve this problem, Ren et al. [25] abandoned the selective search method completely and developed the Region Proposal Network (RPN) algorithm. When RPN and the Fast R-CNN object detector are combined, a new model called Faster R-CNN is developed. Using a RPN algorithm in Faster R-CNN, instead of selective search, cuts down on the number of proposed regions while also ensuring accurate object detection. RPN shares convolutional layers with the object detection networks and it is making proposal computation nearly 10 ms per image. Figure 3.20 illustrates the architecture of the Faster R-CNN model.

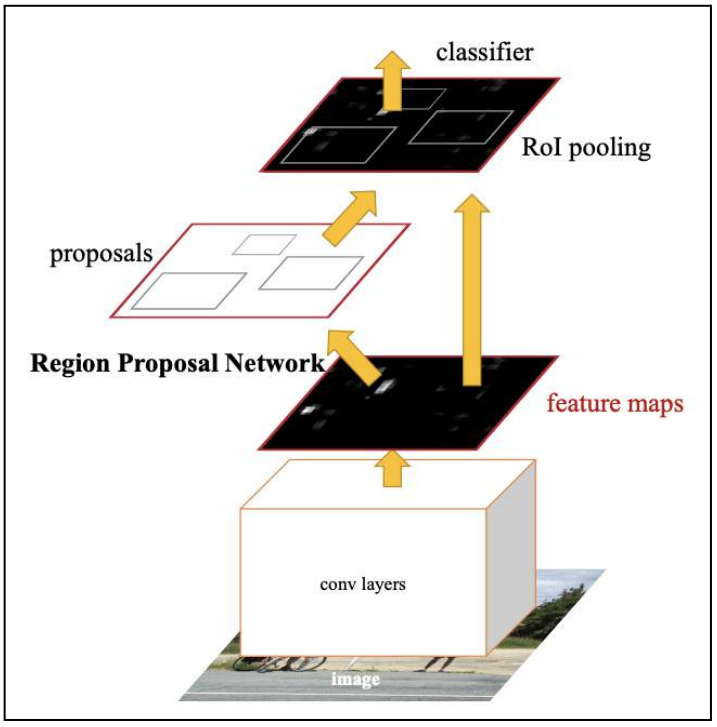


Figure 3.20. The architecture of the Faster R-CNN model [25]

The Faster R-CNN can be evaluated in two stages. In the first stage, RPN is performed which is a deep CNN that is used to generate region proposals. In this stage, RPN instructs the second stage of Faster R-CNN where to search for the object. RPN takes images of any size as input and produces a set of rectangular objects which are proposed as the locations

of the object based on the object score. It makes these proposals by sliding a small spatials window over the feature map generated by the convolutional layer. In the second stage of the Faster R-CNN is the Fast R-CNN object detector. It takes the feature maps that the initial CNN generated and performs RoI pooling on them. Although the layer size varies depending on the size of the input feature map, the output is a fixed feature vector. The RoI output is fed to two fully connected layers which performs classification and regression as in Fast R-CNN. With the introduction of the RPN, Faster R-CNN eliminates the region proposal limitations inherited by R-CNN and Fast R-CNN. Faster R-CNN is also faster in terms of testing time. In this model, the estimation time is reduced to 0.2 seconds.

3.6.1. Region Proposal Network

Region Proposal Network (RPN) architecture [25] computes region proposals in very low time utilizing CNN in an end-to-end learning approach. RPN takes any size image as input and outputs object bounding boxes and objectivity scores simultaneously at each position. RPN is modelled with a fully convolutional network to generate high quality region proposals for object detection using Fast R-CNN. In the model created by combining RPN with Fast R-CNN object detector network as seen in Figure 3.16, RPN shares its computation with Fast R-CNN by sharing a common set of convolutional layers. The image is first passed through the shareable convolutional layers in order to generate proposals for the region in which the object is located. A small network is slid over a convolutional feature map output by the last shared convolutional layer. Following this, each candidate region is mapped to a lower-dimensional feature and this feature is given as an input to two sibling fully connected layers which are the bounding box classification layer and the bounding box regression layer. Accordingly, the classification layer determines the probability of a proposal having the target object. The regression layer adjusts the spatial position of the bounding box. This architecture is naturally implemented with an NxN convolutional layer followed by two sibling 1x1 convolutional layers.

3.6.2. Anchors

RPNs can effectively generate region proposals in a wide variety of scales and with different aspect ratios by using anchor boxes. For this purpose, k region proposals are simultaneously predicted at each sliding window location. The regression layer encodes the coordinates of k boxes by generating 4k outputs and the classification layer outputs 2k scores in order to predict the probability of ‘an object’ or ‘no object’ in each proposal. These k numbers of proposals are referred to as anchors. An anchor is centred at the sliding window. In Faster R-CNN, these anchors are related with three fixed scales (128^2 , 256^2 , and 512^2) and three fixed aspect ratios (1:1, 1:2 and 2:1), resulting in 9 anchor boxes at each sliding region. For a convolutional feature map of a size W×H, there are (W×H)×k anchors in total.

3.7. TESTING METHODOLOGY

In this project, test images are taken in daytime and nighttime to investigate the effect of lighting conditions on model performance. To perform the test more accurately, extra care is taken to ensure that the images are captured at the same place and angle. The photos are taken on the streets of Istanbul's Kadikoy neighbourhood. During the photo shooting, the iPhone 12 Pro's camera is used. Examples of photos taken in the daytime and nighttime are shown in Figure 3.21.

Figure 3.21. Example of photos taken in different lighting conditions. (a) daytime, (b) nighttime.

4. TURKISH TRAFFIC SIGN DATASET CREATION

For a model to be successful, it is critical to train a model with a good and well-designed data set. In order to create a successful model, a new indigenous dataset is required to be generated since previous studies mostly used the German dataset. As part of the thesis, a number of Turkish traffic sign images are collected to train the model, which are obtained from Google maps. While gathering the pictures, it is taken into consideration that an equal amount of pictures are collected for each class to ensure balance in the dataset. The data set contains approximately 200 images from each class. The images are stored in PNG and JPEG formats in various resolutions. The proposed dataset includes 10842 images containing 54 traffic signs. Figure 4.1 illustrates the Turkish Traffic Sign Dataset classes and in Figure 4.2, traffic signs specific to Turkey are shown.

Figure 4.1. Turkish Traffic Sign Dataset in 54 categories

The Turkish Traffic Sign Dataset is created because, even though the German and Turkish traffic signs are similar, there are also distinct signs that are only available in Turkey. The examples are shown in Figure 4.2.

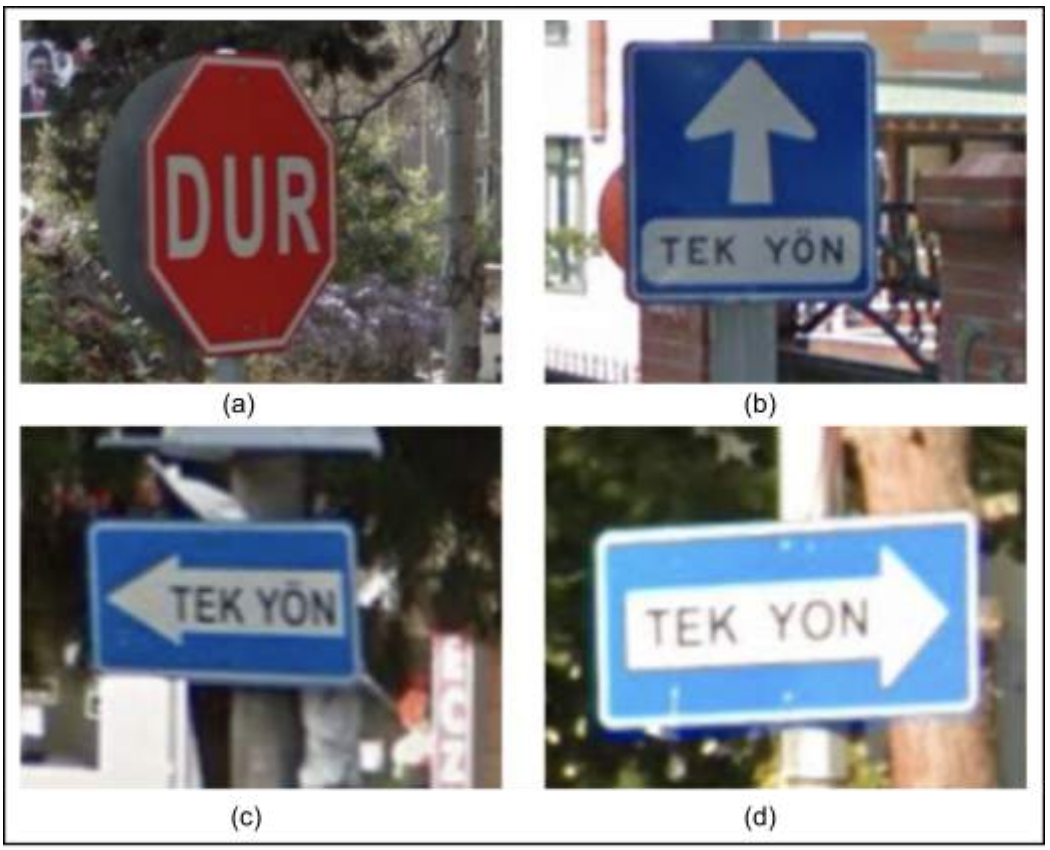


Figure 4.2. Example of traffic signs specific to Turkey. (a) TT-2 traffic sign, (b) B-16 traffic sign, (c) B-16b traffic sign, (d) B-16a traffic sign.

5. UTILIZING SELECTIVE SEARCH AND CNN MODEL: THE IMPACT OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

Since the aim of the thesis is to develop a prototype Turkish TSR system using a newly created Turkish Traffic Sign Dataset, as step one, a CNN model is trained using the selective search method. The model developed in this section is similar to the R-CNN model. After the model training is completed, it is evaluated to investigate the effect of training step numbers, lighting conditions, and image sizes on model performance. The details of the development of the model are explained in detail later in this section. Section 5.1 explains the analysis and design of the system, the system's implementation is described in section 5.2, the evaluation of the developed models and the comparison of their results are described in section 5.3.

5.1. SYSTEM ANALYSIS AND DESIGN

The system analysis and design section, firstly, discusses the programming language, frameworks, and libraries that are used in the realisation of the model in section 5.1.1. Then, Google Colaboratory (Google Colab) is covered in section 5.1.2. Followed by section 5.1.3. that describes transfer learning, and lastly, section 5.1.4. reviews the model architecture.

5.1.1. Programming Language, Frameworks and Libraries

Python programming language is chosen during the development and evaluation of the models. It features a large library and is built on an open source platform. Python is a successful language in terms of performance and simplicity of use due to its modular structure. Choosing the right framework in the development of an object detection model and using this framework in the training of the model is very important for the model to be successful. The neural network framework provides great convenience in the creation of neural networks. These frameworks make model building faster and easier. The

comparison of deep learning frameworks by score is shown in Figure 5.1. Tensorflow and Keras frameworks are used for the model training within the scope of this project.

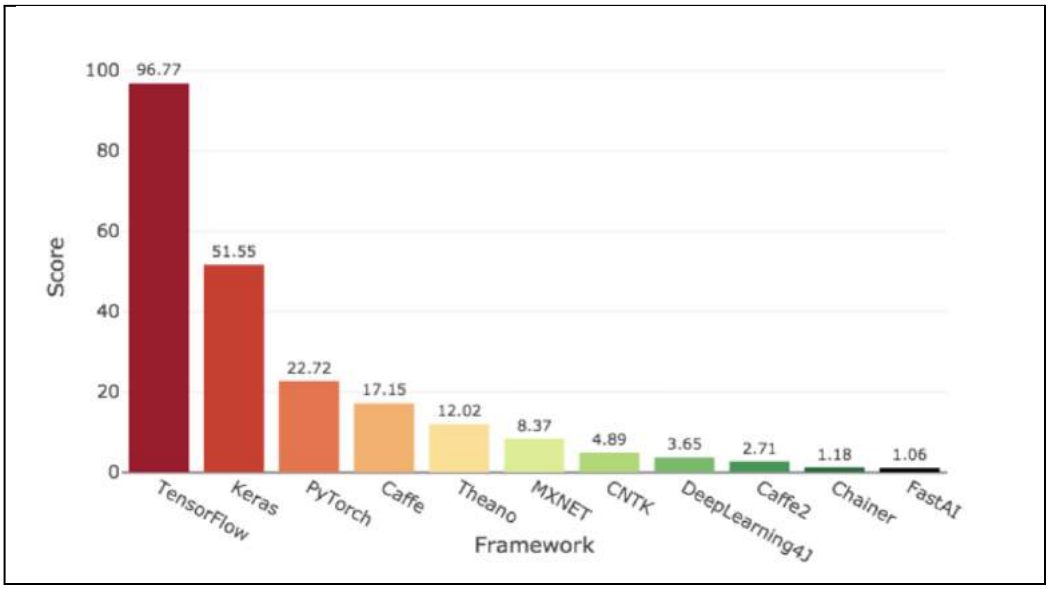


Figure 5.1. Deep learning framework power scores [45]

The frameworks and libraries used during the development of the model are as follows.

TensorFlow [46] is a neural network-focused open source platform built by Google. Its adaptive design allows the process to be deployed to several CPUs or GPUs on a server and no necessitating code rewrites. TensorFlow contains TensorBoard as well. It's a set of tools for visualizing data. Python and C are the two programming languages having TensorFlow APIs that are both stable.

Keras [47] is a Python-based open source neural network library. It is developed with the purpose of enabling rapid experimentation. The three most important features of Keras are that it is simple, flexible and powerful. It supports both convolutional and recurrent networks and operates well on both CPU and GPU.

NumPy [48] is a community-developed, open-source library, used to quickly perform the calculation of scientific data in the Python programming language. The NumPy library allows statistical calculations to be executed with less code when calculating complex calculations on multidimensional arrays.

Matplotlib is a comprehensive library and it is one of Python's most fundamental libraries for data visualization. Two and three dimensional drawings can be created with the help of this library. It's mostly used in two dimensional drawings [49].

OpenCV [50] is a computer vision and machine learning library. It stands for Open Source Computer Vision Library. This library extracts and processes meaningful information from video or image in image processing. It can work independently of many platforms thanks to the functions it contains.

Sklearn [51] is a library for machine learning. It is popular because it is simple to use and has a large, well-designed structure. In this project, the train_test_split function is used to split the dataset into subsets to reduce the possibility of bias in the evaluation and validation process.

Xmltodict [52] is a Python module that facilitates the use of XML. After tagging the traffic signs in the pictures, a file with XML extension is created. It can easily convert XML files to Python dictionaries with its simple function.

5.1.2. Google Colaboratory

After selecting the framework and the pre-trained model, next was the time to decide on which platform the model training should take place. Since the training process of the models takes a lot of time on an average personal computer, it is decided to run these processes on cloud servers. Google Colaboratory [53], or Colab shortly, is used to conduct the training as part of the thesis. Colab is a Jupyter/iPython notebook that allows users to build deep learning applications. Also, this platform supports application development in the python programming language and allows python codes to be run in the browser. Colab provides a completely free use of GPU and TPU. Working with the GPU makes a huge difference in speed compared to the CPU, and the time spent training AI models is much less compared to the CPU. For the reasons stated above, Google Colab is chosen in this project.

5.1.3. Transfer Learning

Transfer learning in machine learning methods refers to storing the knowledge received while solving a problem and utilizing that information when encountered with another problem. Using previous knowledge, this method can build models that are more successful and learn faster with less training data. Learning transfer applies features, weights, etc from previously trained models to a new task. In this project, as step one, a new CNN model is developed using the pre-trained MobileNetV2 model with ImageNet, which has 1000 classes. To achieve this, the fully connected layers of the existing MobileNetV2 network are removed and a new set of fully connected layers is added on top of the CNN. After the new layers are added, the weights are fine-tuned to recognize traffic sign classes. Figure 5.2 illustrates transfer learning with MobileNetV2.

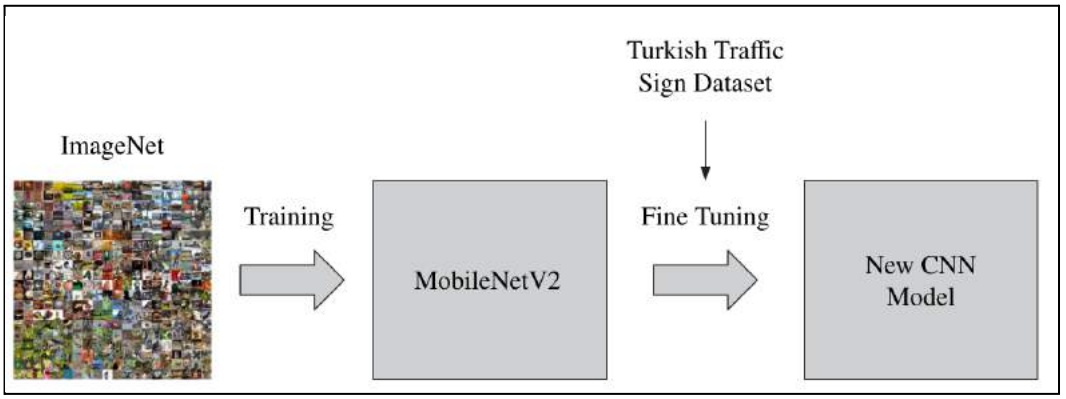


Figure 5.2. Transfer learning with MobileNetV2

5.1.4. Model Architecture

In this chapter, models are developed to detect and recognize traffic signs with the transfer learning method. MobileNetV2 trained on ImageNet is used as the backbone of the newly developed model. The fully connected layers at the end of the network are removed and a new fully connected head is constructed. The previous convolution layers in the network are frozen, and only the FC layer heads are trained. In the network, an average pooling layer with 7x7 pooling regions is added behind the previous convolution layers. After the pooling layer, the data is converted into a 1-dimensional array for input to the following layer by the flattening layer. Following that, two fully connected layers are added to the

network. First one with 128 outputs is added and activated with ReLu. Secondly, since there are 54 classes, a second fully connected layer with output equal to the number of classes is added and the activation function of this layer is softmax. Thus, the top of the CNN model is formed and connected to the MobileNetV2 network's body. Figure 5.3 shows the structure of the CNN model in detail.

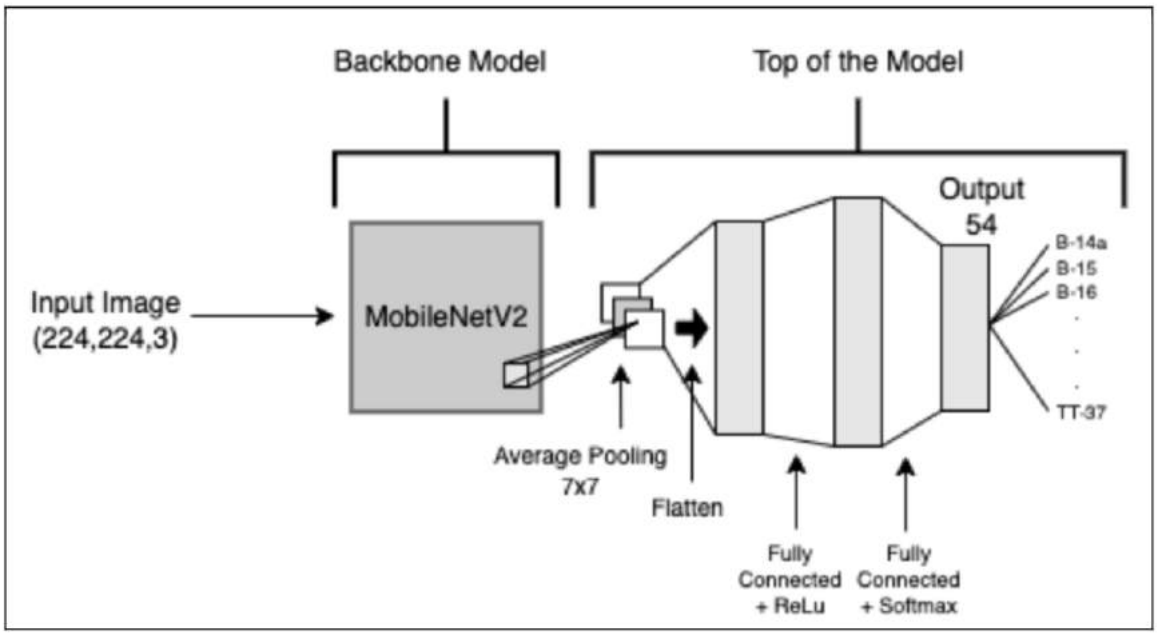


Figure 5.3. The architecture of the CNN model

5.2. IMPLEMENTATION

In this project, as step one, CNN models are trained using the selective search method. The Turkish Traffic Sign Dataset is used while training these models. Using the selective search method, candidate regions, also known as proposals that may contain an object of interest are generated. After the proposals are generated on the images, they are sent into a newly created CNN to produce the actual classifications. Following that, the location and class of the traffic sign in the image are identified by reducing the number of predictions by applying non-maximum suppression to the predictions whose IoU value exceeds the threshold value. The stages of the model implementation are represented in Figure 5.4.

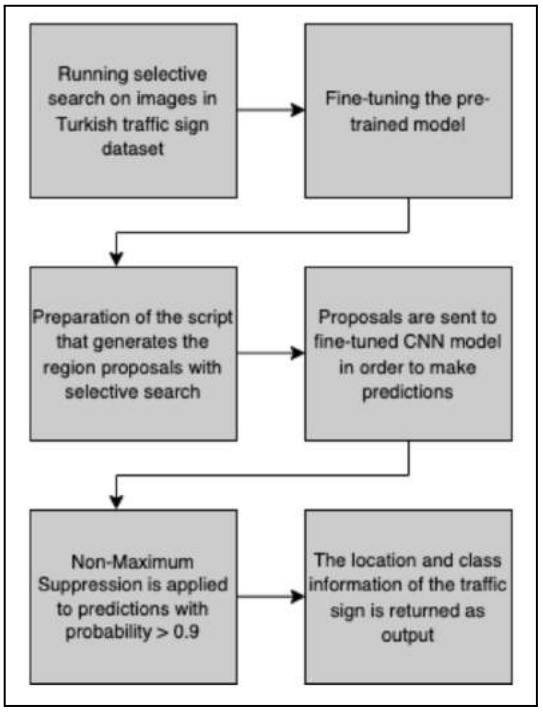


Figure 5.4. The stages of the model implementation

As the first step of the implementation, a dataset of proposals is generated by running the selective search method on the images in the Turkish Traffic Sign Dataset. To accomplish this, the files are arranged into two directories. Images of Turkish traffic signs are located in the Images folder, and XML files of these images are located in the Annotations folder. The ground-truth object detection annotations for the images are stored in XML files. For this reason, the Xmin, Ymin, Xmax, Ymax bounding box coordinates are parsed from the XML files in the annotations folder. Label information is also parsed as well as bounding box coordinates. Xmltodict is used to parse the bounding box and label information in XML files. Following this, to find region proposals on the images, the selective search method is performed and added the results to a list. The IoU calculation is then performed. The region proposals generated by the selective search method and the existing ground truth box information are compared in this computation. Figure 5.5 shows examples of region proposals. The IoU computation is used to identify which regions overlap with the ground truth boxes adequately and which do not. The IoU threshold value is set to 0.7 and the maximum number of proposals is set to 2000. This IoU value is used as a criterion to analyze if a region proposal is a negative or positive RoI. After determining the Negative and Positive RoIs, each RoI is resized to fit CNN input dimensions and stored.

Figure 5.5. Example of region proposals

After preparing the dataset of region proposals, the second stage is to fine-tune the pre-trained CNN model to detect and recognize Turkish traffic signs. The dataset is loaded prior to the training phase, and from there, lists of data and labels are filled via a loop. Then, the data and label lists are converted into NumPy arrays after storing all of the images and their labels in them. The train_test_split() function is utilised with sklearn.model_selection to split data arrays into two subsets. These subsets are used for training and testing. Twenty percent of the total number of images are set aside for testing and the rest for training. To avoid overfitting the model training, the data augmentation technique is performed. Four augmentation operations are used to improve the accuracy of the training models: shifting, zooming, shearing, and rotating images. MobileNetV2 is used as the pre-trained CNN model. To fine-tune the network, it is loaded with pre-trained ImageNet weights, and the network's head is removed. After that, the new head is constructed and it is appended to the network. MobileNet's convolution layers are frozen to prevent it from being trained, and only the newly added FC head is trained. The newly formed network's softmax classifier has 54 outputs that correspond to Turkish traffic sign classes. The Adam optimizer and categorical_crossentropy loss are used to compile the models. The initial learning rate is set to 1e-4 and the batch size is set to 32 before the

training step. To investigate the effect of training step numbers on model success, the training step is repeated twice with different epoch values. The model training process is performed in Google Colab. During the first training, the epoch value is set to 25, and during the second, it is set to 50. After fine-tuning the CNN models, the third stage is to prepare the script that generates region proposals with the selective search. To do this, first the newly created model is loaded, after that, the image containing the traffic signs is loaded, and then perform the selective search on the image. Each proposal is extracted and resized to fit the CNN's input size before being fed to the CNN model for prediction inside a loop created using the region proposal bounding box coordinates generated by selective search. Then RoIs are appended to the proposal list, and the coordinates are appended to the boxes list. Following that, the fourth stage involves feeding the fine-tuned CNN model with proposals in order to classify all of the proposals and make predictions. To do this, the proposals and boxes lists are first converted to numpy arrays. The prediction method is used to feed proposals into the CNN model, and a prediction is returned as an outcome. Minimum probability is set to 0.9. The CNN model's predictions with a probability greater than minimum probability are filtered out. The NMS technique is applied to predictions with a probability greater than minimum probability at the fifth stage. This technique is used to minimize the number of predictions to one. In the sixth stage, after implementing the NMS technique, bounding boxes, labels, and probabilities are displayed on the image.

5.3. EVALUATION & RESULTS

The models trained on newly created Turkish Traffic Sign Dataset are evaluated in order to investigate the effect of training step numbers, lighting conditions, and image sizes on model performance. During the evaluation, the model with an epoch value of 25 is called Model-1.1 and the other model with an epoch value of 50 is called Model-1.2. Turkish traffic signs images are captured in the daytime and nighttime in order to evaluate the models. Different image sizes are also taken into consideration during the evaluation of the developed models. During the evaluation, the maximum number of proposals to be generated with selective search method is set to 2000. The number of proposals to be inferred is also set as 200. The minimum probability used to filter the model's predictions is set to 0.9. The test code for testing the models is written using Python and all evaluations are performed on Google Colab.

Firstly, when the Model-1.1 is evaluated with large size images captured in the daytime and nighttime, Figure 5.6 shows the results.

Figure 5.6. The evaluation of Model-1.1 with the TT-4 large-sized traffic sign

When the TT-4 (no entry) traffic sign is evaluated with Model-1.1, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. However, the model's bounding boxes are drawn to cover more than just the location of the traffic sign both in the daytime and nighttime. Secondly, when the Model-1.1 is evaluated with small size images captured in the daytime and nighttime, Figure 5.7 shows the results.

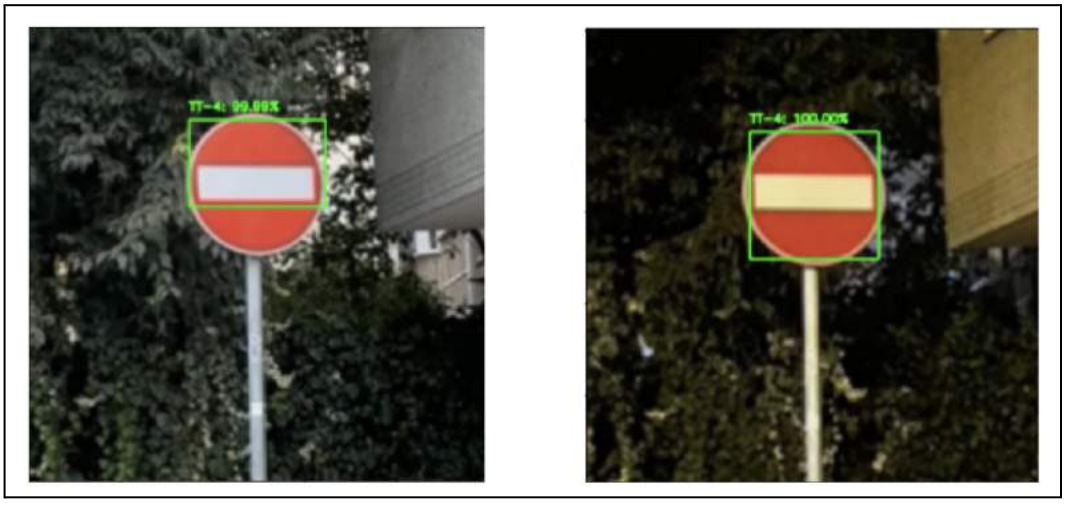


Figure 5.7. The evaluation of Model-1.1 with the TT-4 small-sized traffic sign

Model-1.1 recognized the no entry traffic sign with high accuracy in small size images, both in the daytime and nighttime. Also, the model's bounding box accurately covered the whole traffic sign in the nighttime. It covered more than half of the traffic signs in the daytime. After Model-1.1 is evaluated, Model-1.2 is used to evaluate the no entry traffic sign. Firstly, when the Model-1.2 is evaluated with large size images captured in the daytime and nighttime, Figure 5.8 shows the results.

Figure 5.8. The evaluation of Model-1.2 with the TT-4 large-sized traffic sign

When the no entry traffic sign is evaluated with Model-1.2, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. The model's bounding boxes covered almost the whole traffic sign, leaving only a small area outside the box. Secondly, when the Model-1.2 is evaluated with small size images captured in the daytime and nighttime, Figure 5.9 shows the results.

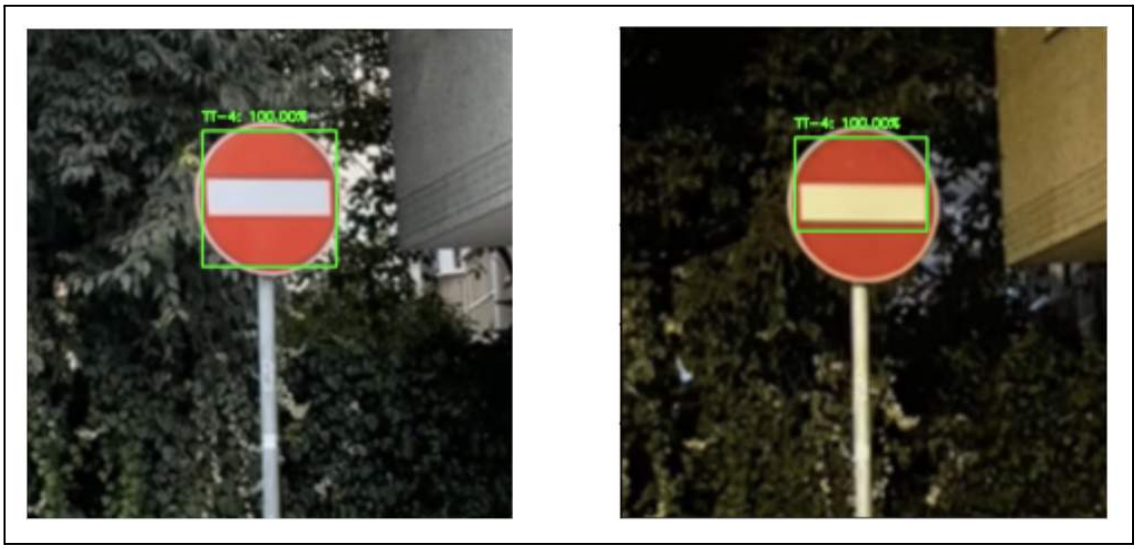


Figure 5.9. The evaluation of Model-1.2 with the TT-4 small-sized traffic sign

Model-1.2 recognized the no entry traffic sign with high accuracy in small size images, both in the daytime and nighttime. Also, the model's bounding box accurately covered the whole traffic sign in the daytime. It covered more than half of the traffic signs in the nighttime. Following the completion of the evaluation of Model-1.1 and Model-1.2 with no entry traffic sign, the evaluation of the models is continued with another traffic sign, B-16 (one way). Firstly, when the Model-1.1 is evaluated with large size images captured in the daytime and nighttime, Figure 5.10 shows the results.

Figure 5.10. The evaluation of Model-1.1 with the B-16 large-sized traffic sign

When the one way traffic sign is evaluated with Model-1.1, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. The model's bounding boxes covered more than half of the traffic sign, leaving only a small area outside the box. Secondly, when the Model-1.1 is evaluated with small size images captured in the daytime and nighttime, Figure 5.11 shows the results. the results.

Figure 5.11. The evaluation of Model-1.1 with the B-16 small-sized traffic sign

Model-1.1 recognized the one way traffic sign with high accuracy in small size images, both in the daytime and nighttime. Furthermore, the model's bounding box covered more than half of the traffic sign both in the daytime and nighttime, as observed in large size images evaluation. After Model-1.1 is evaluated, Model-1.2 is used to evaluate the one way traffic sign. Firstly, when the Model-1.2 is evaluated with large size images captured in the daytime and nighttime, Figure 5.12 shows the results.

Figure 5.12. The evaluation of Model-1.2 with the B-16 large-sized traffic sign

When the one way traffic sign is evaluated with Model-1.2, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. Nevertheless, the model's bounding box covered more than half of the traffic sign both in the daytime and nighttime, as in the Model-1.1 evaluation. Secondly, when the Model-1.2 is evaluated with small size images captured in the daytime and nighttime, Figure 5.13 shows the results.

Figure 5.13. The evaluation of Model-1.2 with the B-16 small-sized traffic sign

Model-1.2 recognized the one way traffic sign with high accuracy in small size images, both in the daytime and nighttime. Also, the model's bounding box accurately covered the whole traffic sign in the nighttime. It covered less than half of the traffic sign in the daytime.

The CNN models as a result of the training are compared using accuracy and loss metrics. When Model-1.1 and Mode-1.2 are compared on accuracy and loss, the accuracy and loss graphs are shown in Figure 5.14. Furthermore, the test score and test accuracy values are compared.

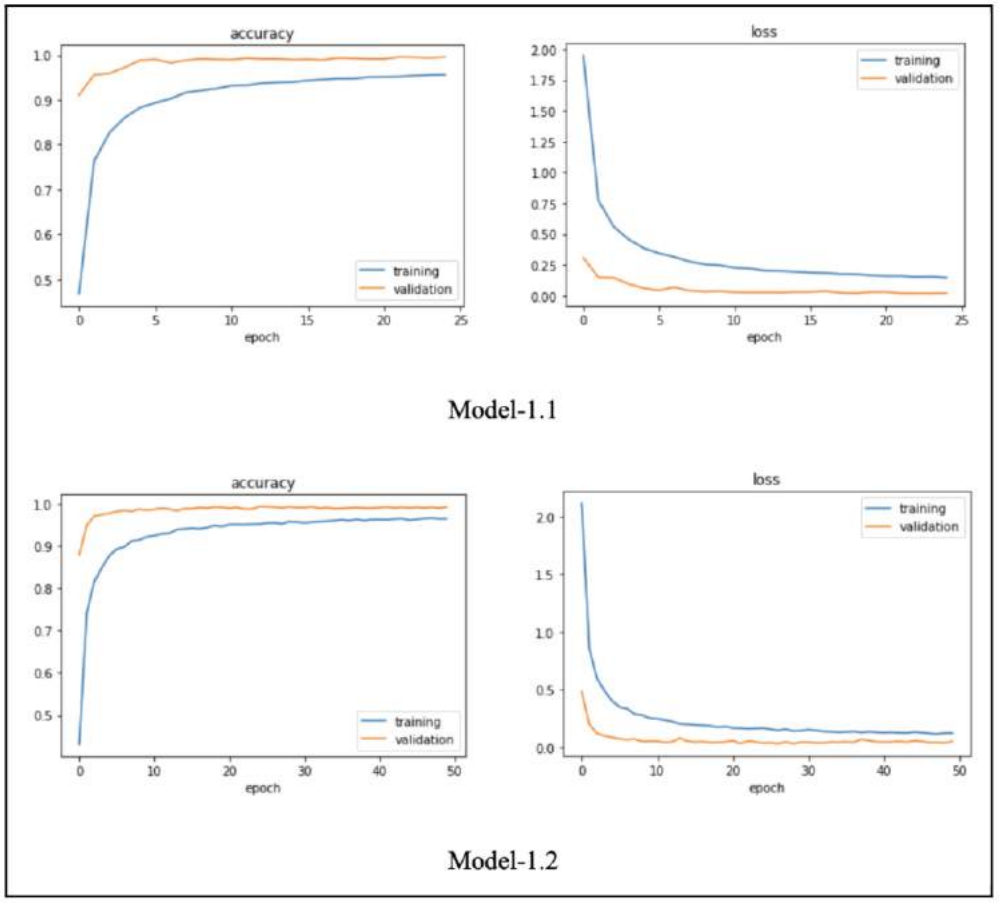


Figure 5.14. Model-1.1 and Model-1.2 accuracy and loss graphs depending epochs

Looking at the graphs of both models, it is clear that Model-1.2 outperforms Model-1.1 in terms of accuracy and also, Model-1.2 reaches lower values than Model-1.1 in terms of loss. The loss value for Model-1.1 is 7 percent, with a test accuracy of 98.7 percent, and the loss value for Model-1.2 is 4 percent, with a test accuracy of 99 percent. These results also support the conclusion that Model-1.2 outperforms Model-1.1. Although the accuracy rate is high, the test time in detection and recognition in both models during evaluation is pretty long. The total test time taken is approximately 51 seconds when the average of all trials performed throughout the evaluation is taken. The majority of the time is spent generating proposals. For this reason, it is not possible to use this system in real time.

6. UTILIZING SELECTIVE SEARCH AND THE CUSTOM CNN MODEL: THE IMPACT OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

In the previous chapter, chapter 5, the models are trained with the transfer learning technique. To fine-tune the network, the fully connected layers of the existing MobileNetV2 network are removed. After that, the new head is constructed and it is appended to the network. MobileNet's convolution layers are frozen to prevent it from being trained, and only the newly added fully connected head is trained. In this chapter, since the aim of the thesis is to develop a prototype Turkish TSR system using a newly created Turkish Traffic Sign Dataset, as step two, a new custom CNN model is developed instead of the transfer learning technique and trained using the selective search method. After the training process, the models developed within the scope of the project are tested with different sized pictures taken in daytime and nighttime and the results are compared. In the continuation of this chapter, how the system is developed and what type of prior preparations are conducted for the system's functioning is detailed. Section 6.1 explains how the system is analyzed and designed, the system's implementation is described in section 6.2, the evaluation of the developed models and the comparison of their results are described in section 6.3.

6.1. SYSTEM ANALYSIS AND DESIGN

The system analysis and design section, firstly, discusses the programming language and libraries that can and are used in the realisation of the custom model in section 6.1.1. It then continues with section 6.1.2, which reviews the model architecture.

6.1.1. Programming Language, Frameworks and Libraries

In this chapter, the Python programming language is used to develop the model and to write the script for model evaluation. Tensorflow and Keras are utilized as frameworks in this chapter, as well as in chapter 5. This chapter also makes use of the NumPy, Matplotlib, OpenCV, Sklearn and Xmltodict libraries, which are also utilized as libraries in chapter 5.

6.1.2. Model Architecture

In Keras, the sequential model is the simplest approach to build a model. The sequential model consists of stacks of layers and these layers can be thought of as the basic blocks that make up the neural network. In this chapter, the network layers are built in a sequential order. First, two convolutional layers are added to the model. These layers have a kernel size of 5x5 and 60 filters. The activation function is the Rectified Linear Unit (ReLu). Behind these two layers, a pooling layer with 2x2 pooling regions is added and max pooling is preferred because it gives better results. After that, two more convolutional layers with a kernel size of 3x3 and 30 filters are added to the model and as the activation function, ReLU is utilized. Then, another pooling layer with 2x2 pooling regions is added behind these two layers. After the second pooling layer, a dropout layer is utilized to keep the model from overfitting. Next, the data is converted into a 1-dimensional array for input to the following layer by the flattening layer. Convolutional layer's output is flattened to generate a single long feature vector. It's also connected to the final classification model, which is referred to as a fully-connected layer. Then a dense layer with 500 outputs is added and activated with ReLu. Finally, since there are 54 classes, a fully connected layer with output equal to the number of classes is added and the activation function of this layer is softmax. Figure 6.1 shows the structure of the custom CNN model in detail.

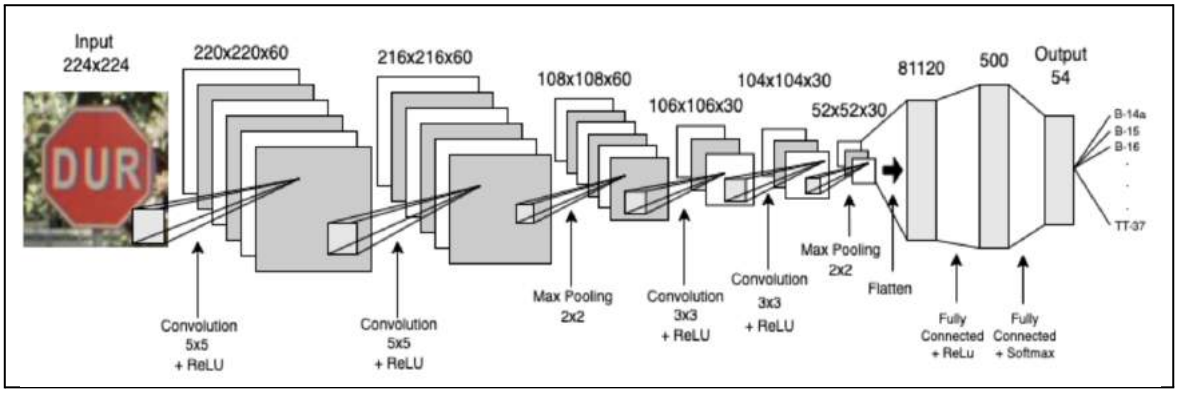


Figure 6.1. The architecture of the custom CNN model

6.2. IMPLEMENTATION

In this chapter, as step two, custom CNN models are trained using the selective search method. The custom CNN models are trained on Google Colab using TensorFlow and Keras frameworks. The model implementation in this chapter follows the same stages as the implementation in Chapter 5. As the first step of the implementation, a dataset of proposals is generated by running the selective search method on the images in the Turkish Traffic Sign Dataset which includes 10842 images containing 54 distinct traffic signs. Figure 6.2 illustrates the training data set distribution.

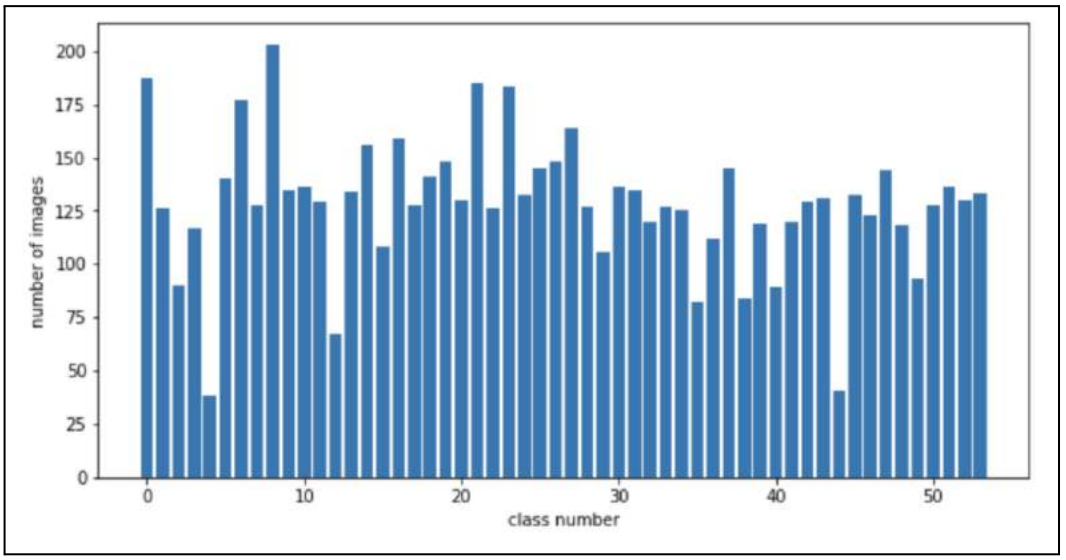
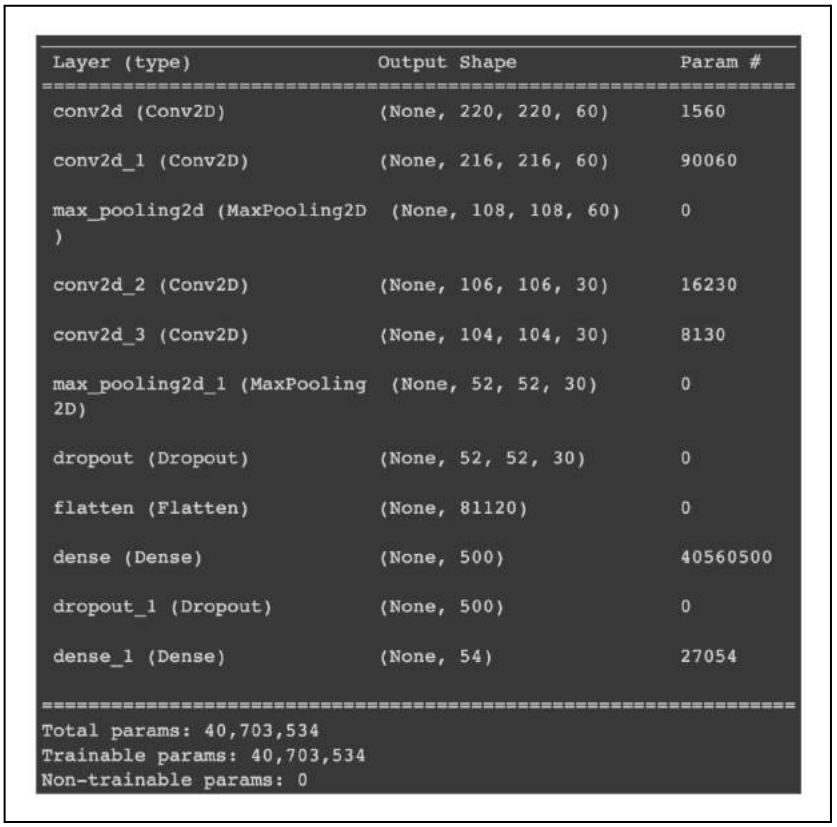


Figure 6.2. The distribution of the training data set

To do this, the files are separated into two folders. As in Chapter 5, the images of Turkish traffic signs are in the images folder, and the XML files of these images are in the annotations folder. To find region proposals on the images, the selective search method is performed and the IoU calculation is then performed. The IoU threshold value is set to 0.7 and the maximum number of proposals is set to 2000. After preparing the dataset of region proposals, the second stage is to develop the custom CNN model to detect and recognize Turkish traffic signs. Figure 6.3 shows the summary of the custom CNN model.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 220, 220, 60)	1560
conv2d_1 (Conv2D)	(None, 216, 216, 60)	90060
max_pooling2d (MaxPooling2D)	(None, 108, 108, 60)	0
conv2d_2 (Conv2D)	(None, 106, 106, 30)	16230
conv2d_3 (Conv2D)	(None, 104, 104, 30)	8130
max_pooling2d_1 (MaxPooling2D)	(None, 52, 52, 30)	0
dropout (Dropout)	(None, 52, 52, 30)	0
flatten (Flatten)	(None, 81120)	0
dense (Dense)	(None, 500)	40560500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 54)	27054

Total params: 40,703,534
Trainable params: 40,703,534
Non-trainable params: 0

Figure 6.3. The summary of the custom CNN model

Using OpenCV, each proposal is resized to 224x224 pixels in order to fit the custom CNN's input size. To be able to feed the developed model with the images as input, the lists are converted into NumPy arrays after storing all of the images and their labels in them. The train_test_split() function is utilized with sklearn.model_selection to split data arrays into two subsets. These subsets are used for training and testing purposes. 20 percent of the total number of images is set aside for testing. Data augmentation is used to enlarge the dataset. This is to prevent model training from becoming overfit. In order to increase the accuracy of the trained models, four augmentation operations are applied, including shifting, zooming, shearing, and rotating images. After that, a custom CNN model is developed. The initial learning rate is set to 1e-4 and the batch size is set to 32 before the training step. After the parameter values are set, the model is trained using the model.fit_generator function. To investigate the effect of training step numbers on model success, the models are trained twice with different epoch values. During the first training, the epoch value is set to 25, and during the second, it is set to 50. Finally, the model.save() function is used to save both of the custom CNN models. After developing the custom

CNN models, the third stage is to prepare the script that generates region proposals with the selective search, as in chapter 5. Then, the selective search method is performed on the image to generate the region proposals. Each proposal is extracted and resized to fit the custom CNN model's input size. After performing the selective search method to the image, the fourth stage involves feeding the newly created custom CNN model with proposals in order to classify all of the proposals and make predictions. Minimum probability is set to 0.9, as in chapter 5. The CNN model's predictions with a probability greater than minimum probability are filtered out. The NMS technique is applied to predictions with a probability greater than minimum probability at the fifth stage and finally, in the sixth stage, after implementing the NMS technique, bounding boxes, labels, and probabilities are displayed on the image. After all these stages are completed, the custom CNN models are ready for evaluation.

6.3. EVALUATION & RESULTS

After being trained with the proposed Turkish Traffic Sign Dataset, the newly trained custom CNN models are evaluated. During the evaluation, the model with an epoch value of 25 is called Model-2.1 and another model with an epoch value of 50 is called Model-2.2. Turkish traffic signs images are captured in the daytime and nighttime in order to evaluate the models. During the examination, specifically traffic signs that are only seen in Turkey and are not included in the GTSDB and GTSRB databases also are evaluated. One Way traffic signs are examples of these. Test images are captured at varying moments of the daytime and nighttime, and extra attention is put in place to ensure that the images are taken in the same spot and at the same angle as mentioned in the previous chapter in order to perform the test with greater accuracy. Different image sizes are also taken into consideration during the evaluation of the developed models. During the evaluation, the maximum number of proposals to be generated with the selective search method is set to 2000. The number of proposals to be inferred is also set as 200. The minimum probability used to filter the model's predictions is set to 0.9. The test code for testing the models is written using Python and all evaluations are performed on Google Colab.

Firstly, when the Model-2.1 is evaluated with large size images captured in the daytime and nighttime, Figure 6.4 shows the results.

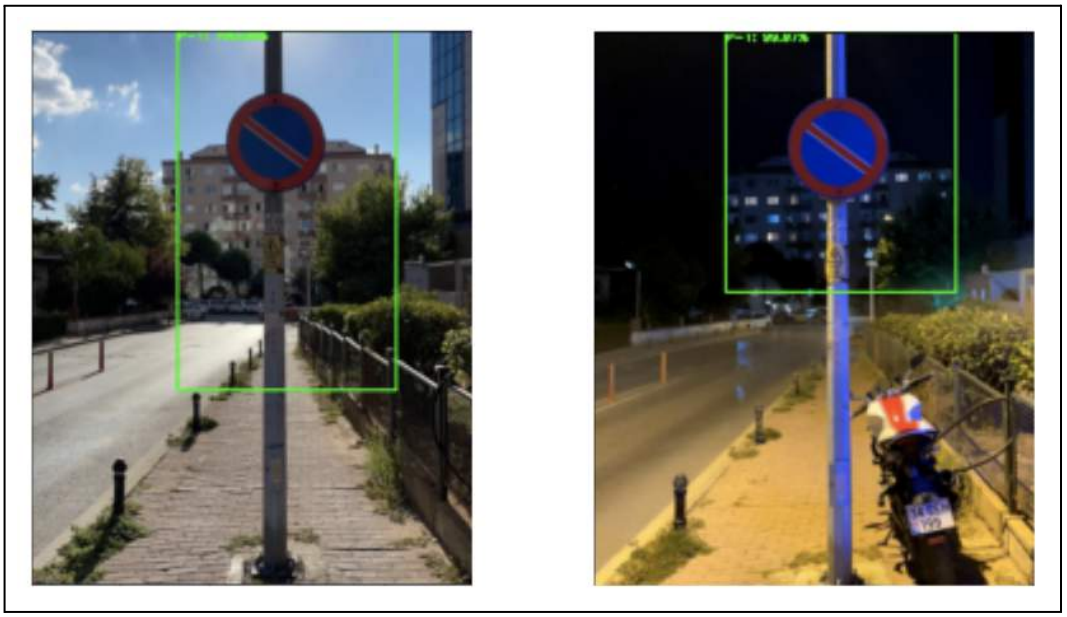


Figure 6.4. The evaluation of Model-2.1 with the P-1 large-sized traffic sign

When the P-1 (no parking) traffic sign is evaluated with Model-2.1, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. However, the model's bounding boxes are drawn to cover more than just the location of the traffic sign both in the daytime and nighttime. Secondly, when the Model-2.1 is evaluated with small size images captured in the daytime and nighttime, Figure 6.5 shows the results.

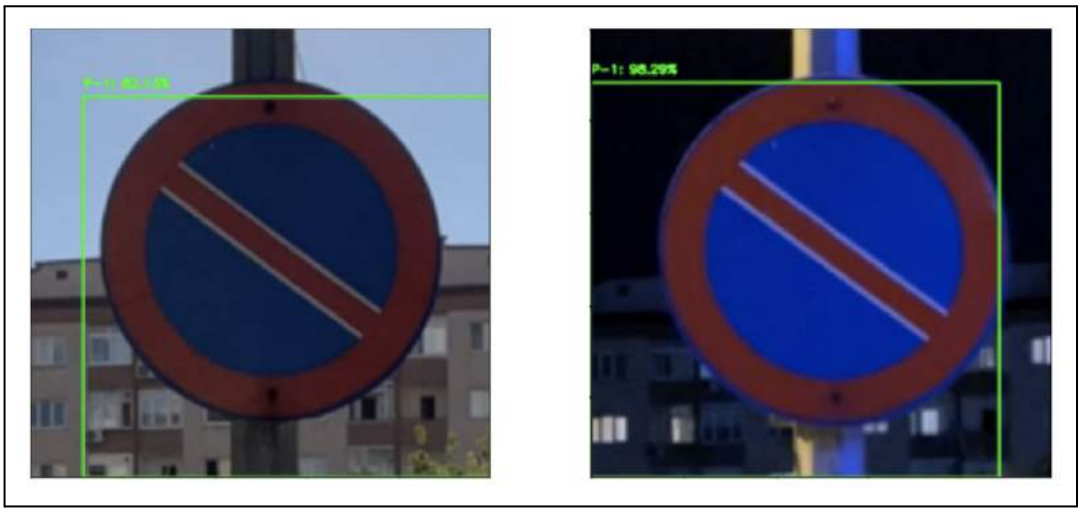


Figure 6.5. The evaluation of Model-2.1 with the P-1 small-sized traffic sign

Model-2.1 recognized the no parking traffic sign with high accuracy on small size images, both in the daytime and nighttime. Also, the model's bounding boxes are drawn to cover more than just the location of the traffic sign both in the daytime and nighttime as in the evaluation with large size images. After Model-2.1 is evaluated, Model-2.2 is used to evaluate the no parking traffic sign. Firstly, when the Model-2.2 is evaluated with large size images captured in the daytime and nighttime, Figure 6.6 shows the results.

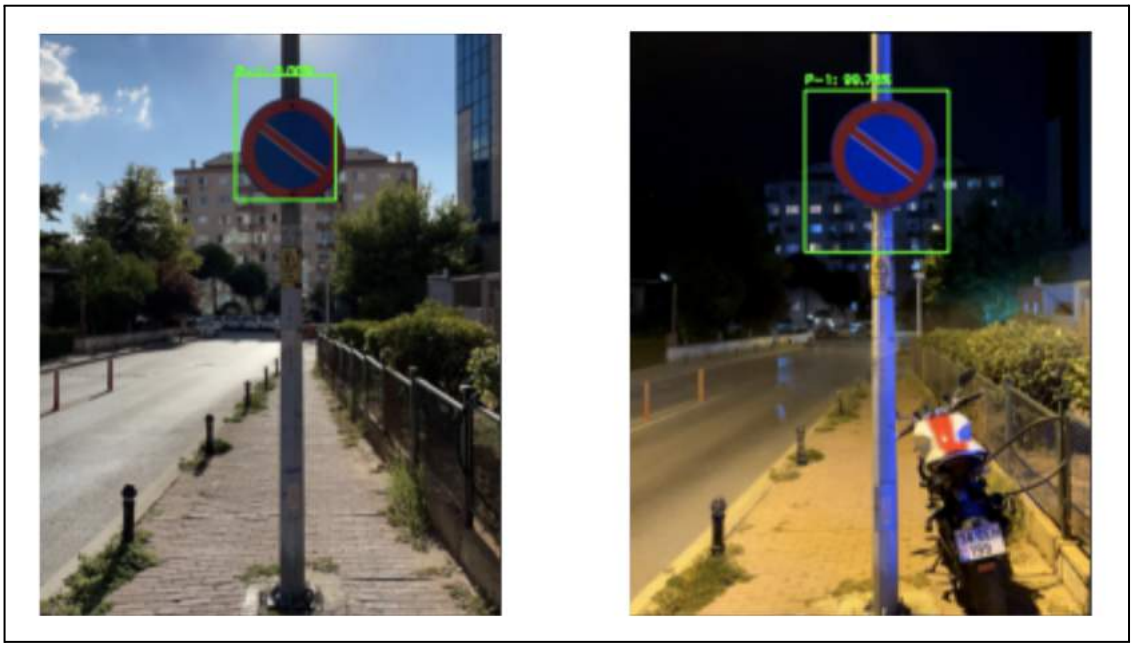


Figure 6.6. The evaluation of Model-2.2 with the P-1 large-sized traffic sign

When the no parking traffic sign is evaluated with Model-2.2, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. The model's bounding boxes covered the whole traffic sign, and the box is drawn to include slightly more area than the traffic sign's location. Secondly, when the Model-2.2 is evaluated with small size images captured in the daytime and nighttime, Figure 6.7 shows the results.

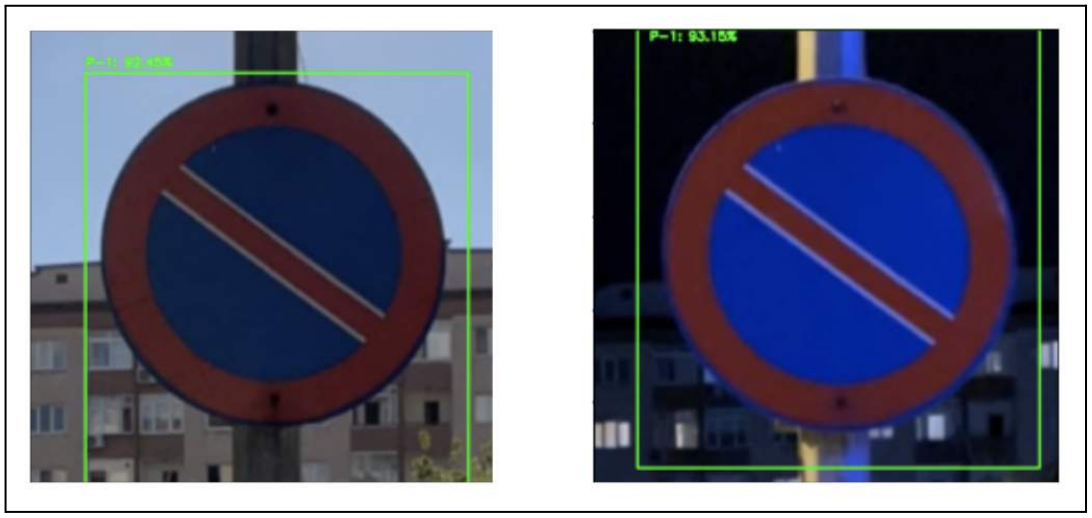


Figure 6.7. The evaluation of Model-2.2 with the P-1 small-sized traffic sign

Model-2.2 recognized the no parking traffic sign with more than 90 percent accuracy on small size images, both in the daytime and nighttime. Also, the model's bounding boxes covered the whole traffic sign, and the box is drawn to include slightly more area than the traffic sign's location both in the daytime and nighttime as in the evaluation with large size images.

Following the completion of the evaluation of Model-2.1 and Model-2.2 with no parking traffic sign, the evaluation of the models is continued with another traffic sign, B-16 (one way). Firstly, when the Model-2.1 is evaluated with large size images captured in the daytime and nighttime, Figure 6.8 shows the results.

Figure 6.8. The evaluation of Model-2.1 with the B-16 large-sized traffic sign

When the one way traffic sign is evaluated with Model-2.1, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. However, the model's bounding boxes are drawn to cover more than just the location of the traffic sign in the daytime. Also, it covered a small area of the traffic sign in the nighttime. Secondly, when the Model-2.1 is evaluated with small size images captured in the daytime and nighttime, Figure 6.9 shows the results. the results.

Figure 6.9. The evaluation of Model-2.1 with the B-16 small-sized traffic sign

Model-2.1 recognized the one way traffic sign with high accuracy in small size images, both in the daytime and nighttime. The model's bounding box covered less than half of the traffic sign in the daytime. It covered more than just the location of the traffic sign in the nighttime. After Model-2.1 is evaluated, Model-2.2 is used to evaluate the one way traffic sign. Firstly, when the Model-2.2 is evaluated with large size images captured in the daytime and nighttime, Figure 6.10 shows the results.

Figure 6.10. The evaluation of Model-2.2 with the B-16 large-sized traffic sign

When the one way traffic sign is evaluated with Model-2.2, it is seen that the traffic sign was recognized by the model with high accuracy both in the daytime and nighttime. The model's bounding box covered less than half of the traffic sign in the daytime. It covered more than just the location of the traffic sign in the nighttime. Secondly, when the Model-2.2 is evaluated with small size images captured in the daytime and nighttime, Figure 6.11 shows the results.

Figure 6.11. The evaluation of Model-2.2 with the B-16 small-sized traffic sign

Model-2.2 recognized the one way traffic sign with high accuracy in small size images, both in the daytime and nighttime. The model's bounding box covered less than half of the traffic sign in the daytime. It covered half of the traffic sign and is drawn to include slightly more area than the traffic sign's location

Accuracy and loss values are used to compare the custom CNN models as a result of the training. Figure 6.12 shows the accuracy and loss graphs when Model-2.1 and Model-2.2 are compared based on accuracy and loss. Additionally, the test score and test accuracy values are also compared.

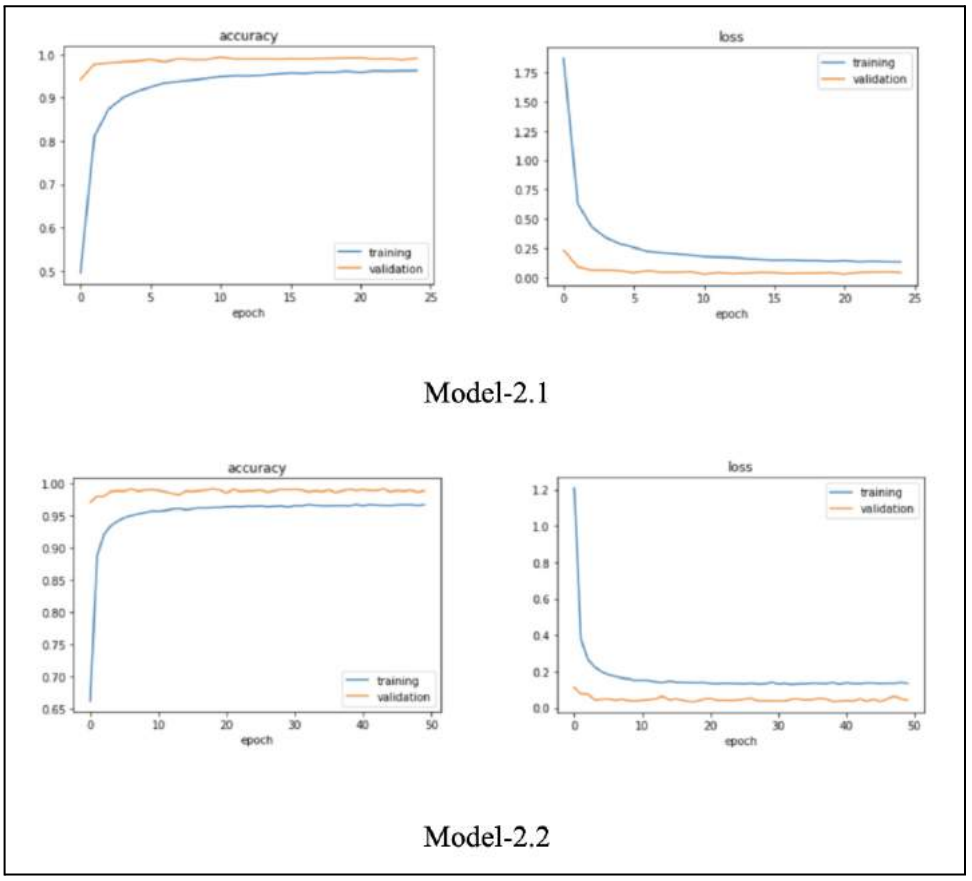


Figure 6.12. Model-2.1 and Model-2.2 accuracy and loss graphs depending epochs

Despite the fact that both models appear to perform similarly in the graphs, Model-2.2 values are better than Model-2.1 in terms of evaluations and results. The loss value for Model-2.1 is 8.9 percent, with a test accuracy of 98.8 percent, and the loss value for Model-2.2 is 9.5 percent, with a test accuracy of 99.1 percent. These results also support the conclusion that Model-2.2 outperforms Model-2.1. Although the accuracy rate is high, the test time in detection and recognition in both models during evaluation is quite long, as in chapter 5. The total test time taken is approximately 63 seconds when the average of all trials performed throughout the evaluation is taken. Thus, it takes longer than the models in Chapter 5. The majority of the test time is spent generating proposals. For this reason, it is not possible to use this system in real time, as in chapter 5.

7. UTILIZING THE FASTER R-CNN MODEL: THE IMPACT OF TRAINING STEP NUMBERS, LIGHTING CONDITIONS AND IMAGE SIZES

In the previous chapters, chapter 5 and chapter 6, the models are trained to develop a prototype Turkish TSR system using a newly created Turkish Traffic Sign Dataset. Both of the models are trained using the selective search method. After the training process, the both models are tested with different sized pictures taken in daytime and nighttime and the results are compared. Despite the fact that the detection and recognition findings are quite accurate, the total test time for both models are quite long. For this reason, it is impossible to use these systems in real time. For these reasons, the Faster R-CNN model is used to develop the Turkish TSR system taking into account the total test time as well as the recognition and detection accuracy. In this chapter, the Faster R-CNN model is analyzed and the model is trained using the Turkish Traffic Sign Dataset with two different training step numbers within the tensorflow framework. After the training process, the models are evaluated to investigate the effect of training step numbers, lighting conditions, and image sizes on model performance and the results are compared.

In the continuation of this chapter, how the system is designed and what kind of preliminary preparations are made for the operation of the system is discussed. Section 7.1 explains how the system is analyzed and designed; section 7.2 describes the steps taken to implement the system, and section 7.3 depicts the evaluation steps of the developed models and analysis of their results.

7.1. SYSTEM ANALYSIS AND DESIGN

Firstly, the Faster R-CNN Inception v2 COCO is discussed in section 7.1.1, followed by a review of the COCO Evaluation Metrics is examined in section 7.1.2.

7.1.1. Faster R-CNN Inception v2 COCO

As explained in the previous chapters, TensorFlow is used as a framework in this chapter. Once a development framework is selected, it is time to select the pre-trained model. TensorFlow is an open source framework that makes it easier to train object detection models. It comes with a large number of pre-trained models. Tensorflow shares these models on its Github page which are referred to as Model Zoo [54]. Because the Faster R-CNN method is utilized in this chapter, pre-trained Faster R-CNN models are reviewed. Following Table 7.1. shows the models in this scope.

Table 7.1. Pre-trained models for Tensorflow [54]

Model Name	Speed (ms)	COCO mAP
faster-rcnn-inception-v2-coco	58	28
faster-rcnn-resnet50-coco	89	30
faster-rcnn-resnet50-lowproposals-coco	64	-
rfcn-resnet101-coco	92	30
faster-rcnn-resnet101-coco	106	32
faster-rcnn-resnet101-lowproposals-coco	82	-
faster-rcnn-inception-resnet-v2-atrous-coco	620	37
faster-rcnn-inception-resnet-v2-atrous-lowproposals-coco	241	-
faster-rcnn-nas	1833	43
faster-rcnn-nas-lowproposals-coco	540	-

The speed values specified in the table are obtained using the Nvidia GeForce GTX TITAN X graphics card [54]. In the COCO mAP column of the table; the higher the value the more successful the model is. Considering the speed and COCO mAP values, the final verdict is to utilize the Faster R-CNN Inception V2 COCO model in this project.

7.1.2. COCO Evaluation Metrics

COCO evaluation metrics [55] are utilized as evaluation criteria for the detection accuracy in this chapter. There are twelve metrics under four main categories in COCO evaluation metrics in Figure 7.1. Average Precision (AP), AP Across Scales, Average Recall (AR), and AR Across Scales are the four main categories.

```
Average Precision (AP):
  AP                % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50        % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75        % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small          % AP for small objects: area < 32^2
  AP^medium         % AP for medium objects: 32^2 < area < 96^2
  AP^large          % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1          % AR given 1 detection per image
  AR^max=10         % AR given 10 detections per image
  AR^max=100        % AR given 100 detections per image
AR Across Scales:
  AR^small          % AR for small objects: area < 32^2
  AR^medium         % AR for medium objects: 32^2 < area < 96^2
  AR^large          % AR for large objects: area > 96^2
```

Figure 7.1. COCO evaluation metrics [55]

The AP and AR are calculated by averaging different IoU values. The IoU threshold in COCO evaluation varies from 0.50 to 0.95 with a 0.05 step size. The AP is calculated by averaging all of the categories. In the area of object detection, this is referred to as mean Average Precision (mAP). The precision and recall are obtained via the following formula in Figure 7.2.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{IoU} = \frac{(Object \cap Detected\ box)}{(Object \cup Detected\ box)}$$

Figure 7.2. The formula of precision, recall and IoU

True positive, false positive, and false negative are denoted by *TP*, *FP*, and *FN*, respectively, in Figure 7.2. The higher the precision and recall, the better the accuracy. The formula for IoU is where Object is the area of the correct object, Detected box is the predicted candidate area, and the area where the two overlap is Object ∩ Detected box. The percentage of image overlap is represented by IoU, which ranges from 0 to 1. The larger the value, the higher the amount of matches between regions, and similarly the lower the value, the less consistency between regions.

7.2. IMPLEMENTATION

In this chapter, the models are trained on Google Colab using Tensorflow with the Faster R-CNN model. Specifically, the Inception V2 COCO pre-trained model is used. The proposed Turkish Traffic Signs Dataset which includes 11353 labels in a total of 10842 images containing 54 traffic signs is used in the training phase. The directory arrangement used while training the models in this chapter is as seen in Figure 7.3.

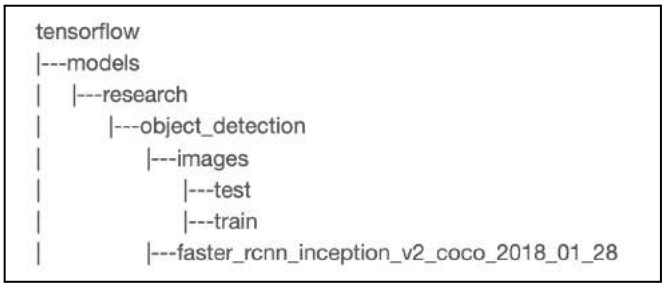


Figure 7.3. The directory structure

Several implementation steps are taken to train and evaluate the model. Figure 7.4 depicts the steps in the model training implementation process.

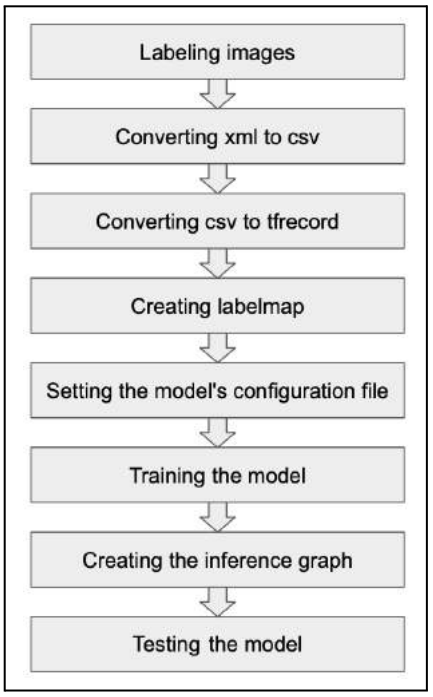


Figure 7.4. The steps of the implementation process

First, the labeling of the images is done with LabelImg, which is a graphical image annotation tool. At this step, the image is opened with the tool and the traffic signs in the image are taken into a square and the traffic sign is associated with the class it belongs to. Following this, the tool creates an XML file. Figure 7.5 shows the name of this XML file is the same as the image. The content of the XML file is shown in Figure 7.6.

Ad	Değişiklik Tarihi	Büyüklük	Tür
Ekran Resmi 2021-03-14 11.05.55.xml	14 Ağustos 2021 12:57	597 bayt	XML Document
Ekran Resmi 2021-03-14 11.08.26	14 Mart 2021 11:08	809 KB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.08.26.xml	14 Ağustos 2021 12:58	595 bayt	XML Document
Ekran Resmi 2021-03-14 11.08.33	14 Mart 2021 11:08	1,1 MB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.08.33.xml	14 Ağustos 2021 12:58	596 bayt	XML Document
Ekran Resmi 2021-03-14 11.19.57	14 Mart 2021 11:20	1,4 MB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.19.57.xml	14 Ağustos 2021 12:58	597 bayt	XML Document
Ekran Resmi 2021-03-14 11.20.04	14 Mart 2021 11:20	712 KB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.20.04.xml	14 Ağustos 2021 12:59	596 bayt	XML Document
Ekran Resmi 2021-03-14 11.21.12	14 Mart 2021 11:21	1,6 MB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.21.12.xml	14 Ağustos 2021 12:59	597 bayt	XML Document
Ekran Resmi 2021-03-14 11.21.22	14 Mart 2021 11:21	1,3 MB	PNG görüntüsü
Ekran Resmi 2021-03-14 11.21.22.xml	14 Ağustos 2021 12:59	596 bayt	XML Document
Ekran Resmi 2021-03-14 21.38.43	14 Mart 2021 21:38	1 MB	PNG görüntüsü
Ekran Resmi 2021-03-14 21.38.43.xml	14 Ağustos 2021 13:00	596 bayt	XML Document
Ekran Resmi 2021-03-14 21.50.57	14 Mart 2021 21:51	445 KB	PNG görüntüsü

Figure 7.5. Prepared XML files with LabelImg

```
<annotation>
        <folder>TT-26b</folder>
        <filename>IMG_6411.JPG</filename>
        <path>/Users/kaankocakanat/Desktop/KAAN THESIS/Test/TT-26b/IMG_6411.JPG</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>3024</width>
                <height>4032</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>TT-26b</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>613</xmin>
                        <ymin>1747</ymin>
                        <xmax>792</xmax>
                        <ymax>1921</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 7.6. The content of XML file prepared with LabelImg

Following the labeling process, the dataset is splitted into two folders, training and test, at a ratio of approximately 80:20 under the images folder as can be seen in Figure 7.7. This folder contains 8672 pictures with 9114 labels are used as train data, and 2170 pictures with 2241 labels are used as test data.

Figure 7.7. Separation of images into two groups as test and train

Then, the XML files are converted to CSV files with a script named xml_to_csv.py written in python. Then, it would generate two files in the images folder called train_labels.csv and test_labels.csv. The contents of the CSV file are shown in Figure 7.8.

filename	width	height	class	xmin	ymin	xmax	ymax
Ekran Resmi 2021-08	496	800	T-22b	95	230	131	260
00008_00000_0001	42	44	TT-29-120	5	7	36	41
Ekran Resmi 2021-05	410	540	TT-1	26	144	313	424
Ekran Resmi 2021-08	180	178	T-22b	48	33	144	161
00008_00011_0002	63	62	TT-29-120	8	8	55	54
Ekran Resmi 2021-05	144	152	T-22b	8	19	134	131
Ekran Resmi 2021-03	500	386	TT-1	65	31	435	367
Ekran Resmi 2021-05	184	178	TT-29-20	29	22	162	167
Ekran Resmi 2021-05	368	292	T-16	63	44	325	276
Ekran Resmi 2021-05	154	200	TT-35d	18	17	139	185
Ekran Resmi 2021-05	74	94	TT-29-80	1	14	71	87
Ekran Resmi 2021-03	140	134	TT-35a	45	29	105	102
Ekran Resmi 2021-05	370	540	T-20	86	60	299	249
Ekran Resmi 2021-05	90	84	T-22c	7	13	73	71
Ekran Resmi 2021-03	142	216	TT-4	7	13	125	203
Ekran Resmi 2021-08	496	802	T-2b	33	284	95	339
Ekran Resmi 2021-05	370	540	T-22a	132	148	210	225
00007_00029_0002	55	56	TT-29-100	4	5	49	51
Ekran Resmi 2021-05	370	540	T-8	109	318	145	350
Ekran Resmi 2021-05	116	152	B-15	26	18	99	137

Figure 7.8. The content of CSV file

TFRecords files are generated with generate_tfrecords.py using train_labels.csv and test_labels.csv once the XML files are converted to CSV files. Then, train.records and test.records files are generated. Figure 7.9 shows the content of the generate_tfrecords.py

```
def class_text_to_int(row_label):
    if row_label == 'B-14a':
        return 1
    elif row_label == 'B-15':
        return 2
    elif row_label == 'B-16':
        return 3
            .
            .
            .
    elif row_label == 'TT-37':
        return 52
    elif row_label == 'TT-4':
        return 53
    elif row_label == 'TT-5':
        return 54
    else:
        None
```

Figure 7.9. The content of generate_tfrecords.py

Following the CSV files are converted to TFRecords file, the label map is required which maps labels to IDs. A labelmap is created with the name and id information of each class to be detected and recognized by the model. The labelmap.pbtxt file is shown in Figure 7.10.

```
item {
  id: 1
  name: 'B-14a'
}
item {
  id: 2
  name: 'B-15'
}
    .
    .
item {
  id: 53
  name: 'TT-4'
}

item {
  id: 54
  name: 'TT-5'
}
```

Figure 7.10. The content of labelmap.pbtxt

After the label map is created, num_classes in the model's config file is changed to 54, since there are 54 classes to be recognized. The num_examples value in the model's config file has been updated to 2170 because there are 2170 test data in the model evaluation. Then, the num_steps value in the model's config file is changed to 51,217 (Model-3.1) in the first training and 200,000 (Model-3.2) in the second training. These models are trained using Python programming language and Tensorflow version 1.15.2 on the Nvidia Tesla K80 graphic card. They are used to train using 51,217 and 200,000 steps, respectively. It took approximately 2 hours to train with 51,217 steps, while it took over 8 hours with 200,000 steps. A frozen inference graph is generated in order to use the models after training. After this process is done, the models are ready to be used for testing.

7.3. EVALUATION & RESULTS

After the model training is carried out with the proposed Turkish Traffic Sign Dataset using the Faster R-CNN Inception V2 COCO model, the newly trained two models are put to the test. In order to carry out model analyses, pictures of Turkish traffic signs are collected. Some of these signs are specifically selected since they are exclusively seen in Turkish traffic signs and are not included in the datasets of GTSDB and GTSRB - traffic signs such as Stop and One Way. Test images are taken in the daytime and nighttime. In order to perform the test with more accuracy, extreme attention is paid so that the images are taken at the same place and at the same angle. The test code for testing the models is written using Python and executed on Google Colab. Firstly, when the Model-3.1 and Model-3.2 are evaluated with large size images captured in the daytime and nighttime, Figure 7.11 shows the results.

Figure 7.11. Recognition trials of the B-14a and the TT-2 large-sized traffic signs in the daytime and nighttime with Model-3.1 and Model-3.2

Model-3.1 is evaluated with the B-14a (Pedestrian Crossing) and the TT-2 (Stop) traffic signs in the daytime and nighttime. The both traffic signs are recognized by the Model-3.1 as can be seen in Figure 7.11. Model-3.1 recognized the B-14a with 99 percent accuracy and the TT-2 with 97 percent accuracy in the nighttime. Also, it recognized the B-14a with 99 percent accuracy and the TT-2 with 95 percent accuracy in the daytime. When Model-3.2 is evaluated with the same traffic signs and under the same conditions, it recognizes both traffic signs with high accuracy. Model-3.2 recognized the B-14a with 100 percent accuracy and the TT-2 with 99 percent accuracy in the nighttime. It recognized the B-14a and the TT-2 with 99 percent accuracy in the daytime. Secondly, when the Model-3.1 and Model-3.2 are evaluated with small size images captured in the daytime and nighttime, Figure 7.12 shows the results.

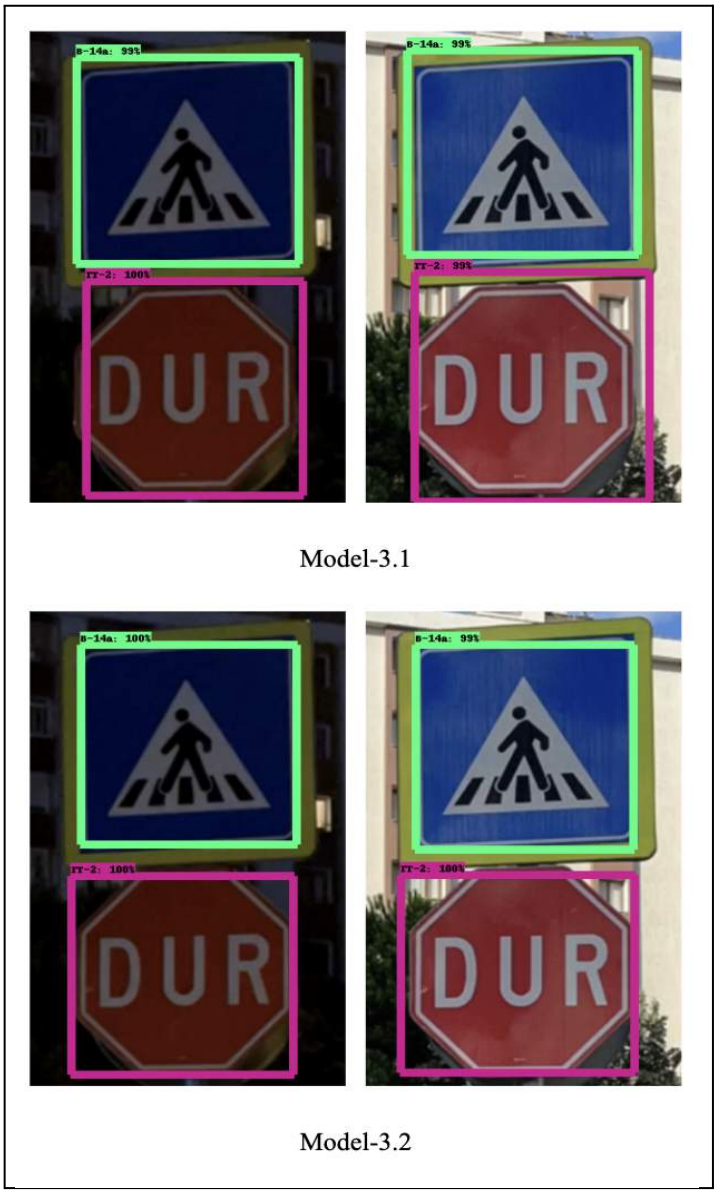


Figure 7.12. Recognition trials of the B-14a and the TT-2 small-sized traffic signs in the daytime and nighttime with Model-3.1 and Model-3.2

The both traffic signs are recognized by the Model-3.1 as can be seen in Figure 7.12. Model-3.1 recognized the B-14a with 99 percent accuracy and the TT-2 with 100 percent accuracy in the nighttime. Also, it recognized the B-14a with 99 percent accuracy and the TT-2 with 99 percent accuracy in the daytime. When Model-3.2 is evaluated with the same traffic signs and under the same conditions, it recognizes both traffic signs with high accuracy. Model-3.2 recognized the B-14a with 99 percent accuracy and the TT-2 with 100 percent accuracy in the daytime. It recognized the B-14a and the TT-2 with 100 percent accuracy in the nighttime.

Model-3.1 and Model-3.2 are evaluated in another environment using another traffic sign, B-16 (One Way). Since the B-16 contains Turkish character, it is critical for evaluation. Firstly, when the Model-3.1 and Model-3.2 are evaluated with large size images captured in the daytime and nighttime, Figure 7.13 shows the results. Model-3.1 recognized the B-16 with 97 percent accuracy in the daytime. However, it could not recognize the B-16 in the nighttime. Unlike the Model-3.1, the Model-3.2 recognized the B-16 both in the daytime and nighttime when evaluated with the same traffic sign under the same conditions. Model-3.2 recognized the B-16 with 92 percent accuracy in the nighttime and 99 percent accuracy in the daytime.

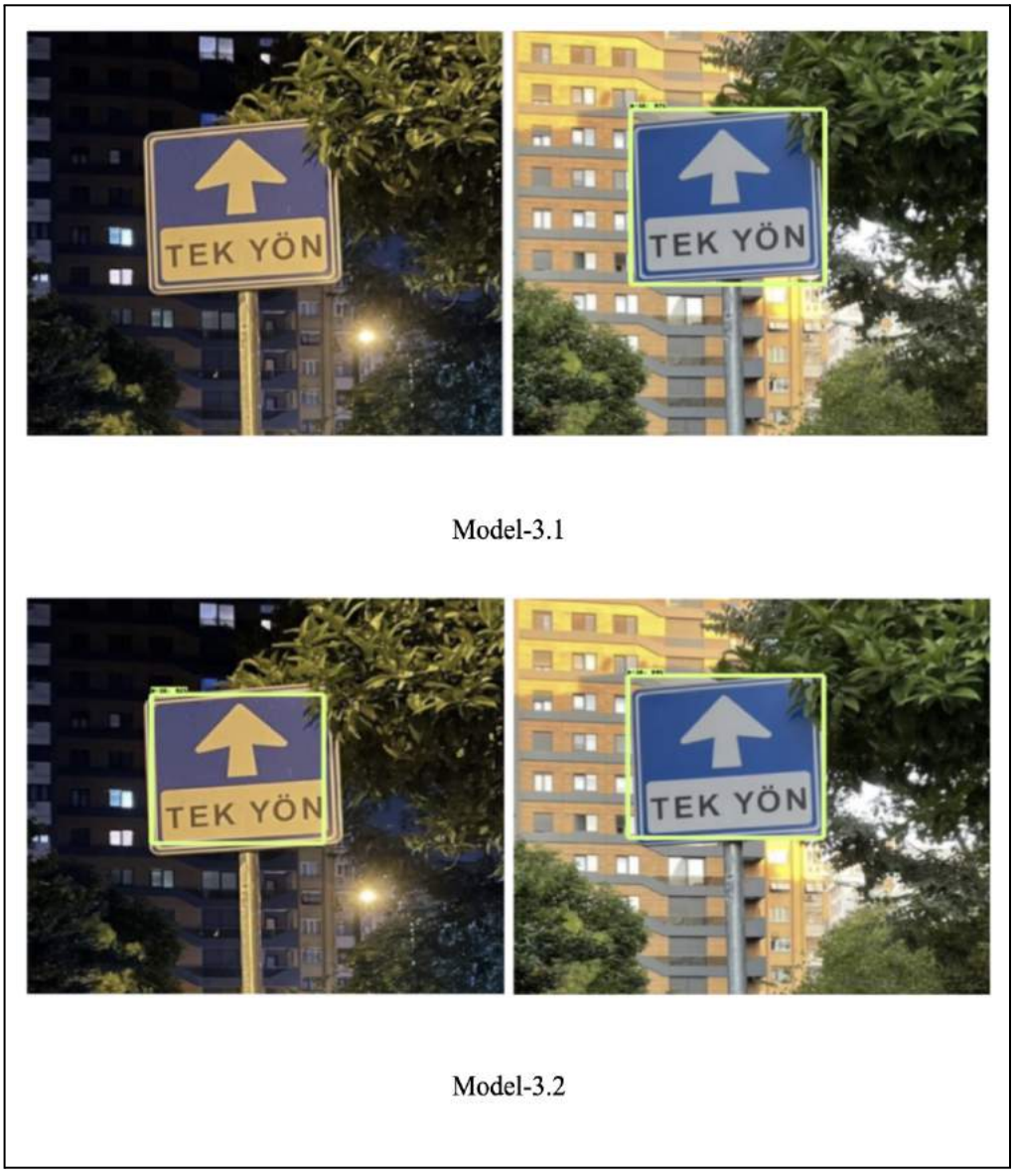


Figure 7.13. Recognition trials of the B-16 large-sized traffic sign in the daytime and nighttime with Model-3.1 and Model-3.2

Secondly, when the Model-3.1 and Model-3.2 are evaluated with small size images captured in the daytime and nighttime, Figure 7.14 shows the results.

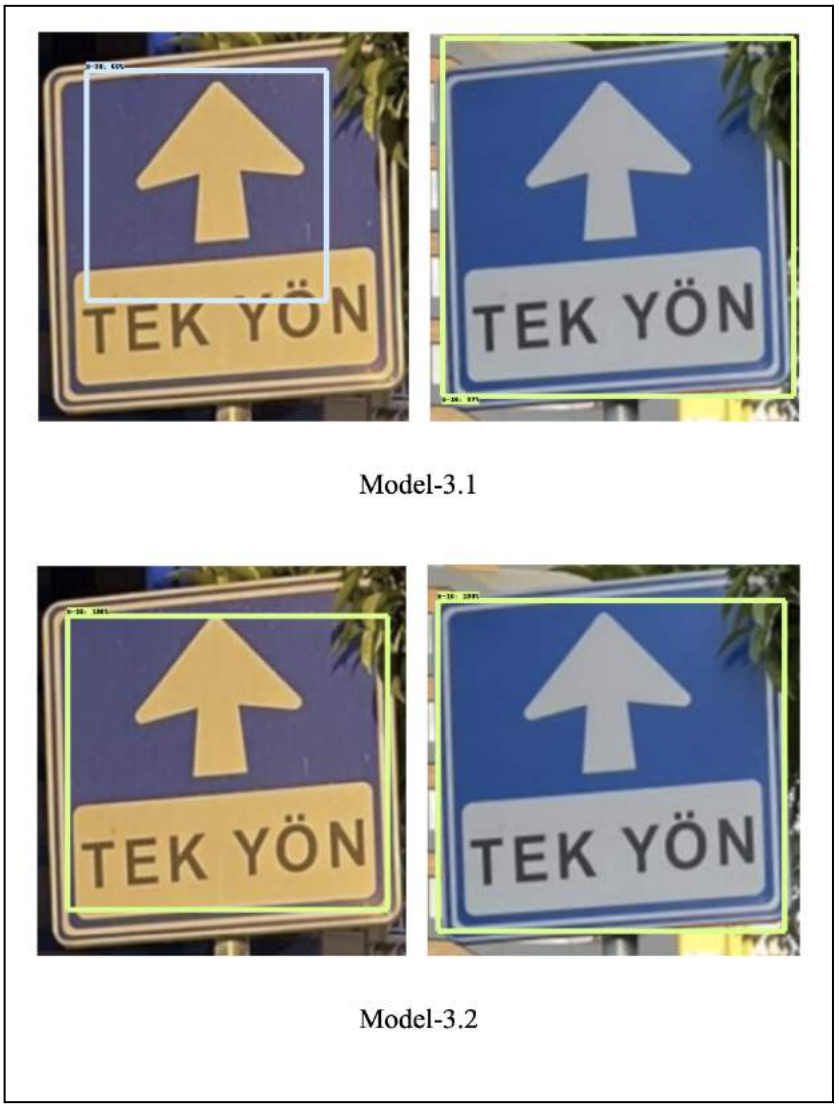


Figure 7.14. Recognition trials of the B-16 small-sized traffic signs in the daytime and nighttime with Model-3.1 and Model-3.2

The B-16 traffic sign is recognized by the Model-3.1 as can be seen in Figure 7.14. Model-3.1 recognized the B-16 with 68 percent accuracy and covered less than half of the traffic sign in the nighttime. Also, it recognized the B-16 with 97 percent accuracy in the daytime. When Model-3.2 is evaluated with the same traffic signs and under the same conditions, it recognizes both traffic signs with high accuracy. Model-3.2 recognized the B-16 with 100 percent accuracy both in the daytime and nighttime.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.672
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.851
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.832
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.490
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.683
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.691
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.783
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.785
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.785
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.583
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.770
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.800
```

Figure 7.15. Average precision and average recall values of the Model-3.1

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.760
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.903
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.893
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.554
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.763
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.778
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.828
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.828
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.828
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.617
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.818
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.844
```

Figure 7.16. Average precision and average recall values of the Model-3.2

The COCO metrics are used to compare the AP and AR values of the models that as a result of the training. Model-3.1 provided the results shown in Figure 7.15, while Model-3.2 provided the results shown in Figure 7.16. These two models are evaluated on 2170 test data. During the evaluation, in order to assess the model's success, AP and AR values are found and averaged for 10 different threshold values, starting from 0.5 and going up to 0.95 with 0.05 increments. Figures 5.15 and 5.16 demonstrate, AP is 67.2 percent for Model-3.1 and 76 percent for Model-3.2. AR is 78.3 percent for Model-3.1 and 82.8 percent for Model-3.2. As a result of the comparison, it is obtained that Model-3.2 is more successful than Model-3.1 in terms of average precision and average recall. Furthermore, loss values are used to compare the custom CNN models as a result of the training. Figure 7.17 represents the loss values for the Model-3.1 and the Model-3.2 according to the training step numbers.

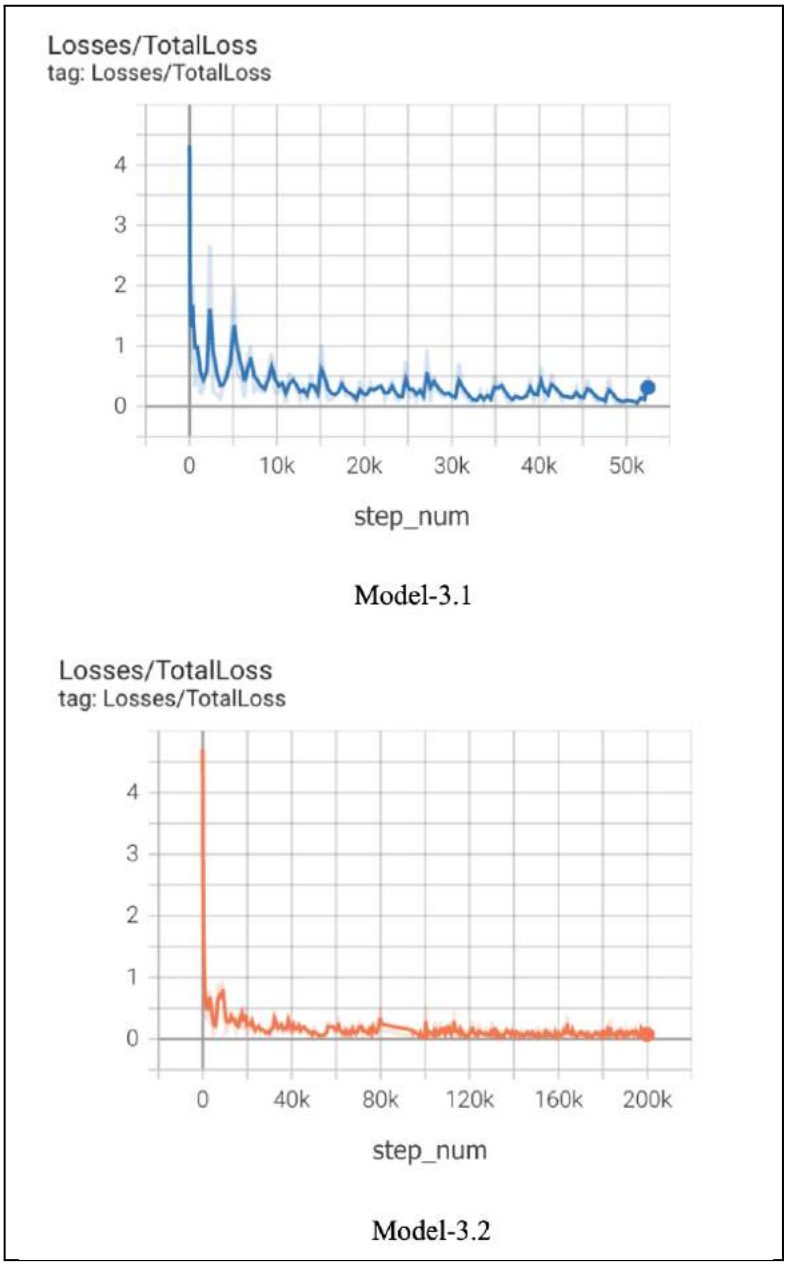


Figure 7.17. Loss values for the Model-3.1 and the Model-3.2 according to step_num

The loss value for Model-3.1 is 4 percent and for Model-3.2 is 1.9 percent. These results also support the conclusion that Model-3.2 outperforms Model-3.1. At the same time, the test time of the Faster R-CNN model is very short when compared to the models in previous chapters. When the average of all trials performed throughout the evaluation is taken, the total test time is approximately 7 seconds. Although it provides high accuracy in a short test time, this system is still considered slow to be used in real time.

8. OVERALL EVALUATION AND COMPARISON OF THE RESULTS

This chapter provides an overall evaluation and comparison of the models developed as part of the thesis. In order to develop a Turkish TSR system, three different steps are carried out in this thesis. The training process of the models are carried out twice with different training step numbers. Then, these models are used to detect Turkish traffic sign images taken both daytime and nighttime with large size and small size images. Firstly, as step one, both models are able to detect and recognize the Turkish traffic signs with high accuracy in small and large size images, both in the daytime and nighttime. Model-1.2 outperforms Model-1.1 in terms of accuracy and also, Model-1.2 reaches lower values than Model-1.1 in terms of loss. Secondly, as step two, both models are able to detect and recognize the Turkish traffic signs with high accuracy in small and large size images, both in the daytime and nighttime, as in step one. Although the loss value of Model-2.2 is higher than Model-2.1, its accuracy is also higher than Model-2.1. Furthermore, Model-2.2 values are better than Model-2.1 in terms of evaluations and results. Lastly, as step three, The COCO metrics are used to compare the average precision and average recall values of the models that as a result of the training. Model-3.2 outperforms Model-3.1 in terms of average precision and also, Model-3.2 reaches higher values than Model-3.1 in terms of average recall. As a result of the comparison, it is obtained that Model-3.2 is more successful than Model-3.1. Despite the fact that the models in all three steps detect and recognize Turkish traffic signs with high accuracy, the total test time during the evaluations differ. When the total test time for all three steps are compared, the detection and recognition time for the models are approximately 51, 63, and 7 seconds, respectively. Table 8.1 illustrates the performance comparison of the models in each of the three steps for the B-16 traffic sign and Figure 8.1 compares total test time for each step.

Table 8.1. Performance comparison of the models for the B-16 traffic sign

Traffic Sign	Step One		Step Two		Step Three	
B-16	Model-1.1	Model-1.2	Model-2.1	Model-2.2	Model-3.1	Model-3.2
	 Accuracy: 100%	 Accuracy: 100%	 Accuracy: 100%	 Accuracy: 99.99%	 Accuracy: 97%	 Accuracy: 99%
	 Accuracy: 100%	 Accuracy: 100%	 Accuracy: 99.99%	 Accuracy: 100%	 Accuracy: -	 Accuracy: 92%
	 Accuracy: 99.99%	 Accuracy: 99.52%	 Accuracy: 100%	 Accuracy: 99.94%	 Accuracy: 97%	 Accuracy: 100%
	 Accuracy: 100%	 Accuracy: 100%	 Accuracy: 97.72%	 Accuracy: 100%	 Accuracy: 68%	 Accuracy: 100%

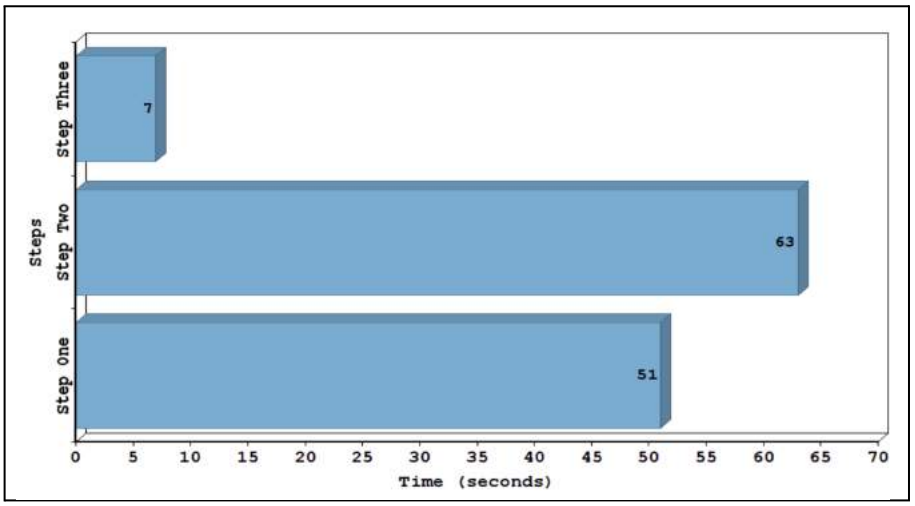


Figure 8.1. Comparison of total test time for each step

9. CONCLUSION AND FUTURE WORK

This thesis follows three distinct steps in order to develop a Turkish TSR system. Firstly, as step one, models are trained using the selective search method. In step one, transfer learning is utilized in model training and models are developed using the pre-trained MobileNetV2 model. Secondly, as step two, models are trained using the selective search method once more, however this time instead of using the transfer learning, the models are trained by developing a new custom CNN. Lastly, as step three, the Faster R-CNN Inception V2 COCO model is utilized in model training. As part of the thesis, the Turkish Traffic Sign Dataset is created by collecting the images of the Turkish traffic signs and used in the training of models. Models are trained twice, each time with a different number of training step numbers. These models are then used to detect and recognize Turkish traffic sign images collected during the daytime and nighttime, in large and small sizes. In step one, the loss value for Model-1.1 is 7 percent, with a test accuracy of 98.7 percent and the loss value for Model-1.2 is 4 percent, with a test accuracy of 99 percent. In step two, the loss value for Model-2.1 is 8.9 percent, with a test accuracy of 98.8 percent and the loss value for Model-2.2 is 9.5 percent, with a test accuracy of 99.1 percent. Lastly, in step three, average precision is 67.2 percent for Model-3.1 and 76 percent for Model-3.2. Average recall is 78.3 percent for Model-3.1 and 82.8 percent for Model-3.2. Also, the loss value for Model-3.1 is 4 percent and for Model-3.2 is 1.9 percent. Although the accuracy rate is high, the time spent in detection and recognition in steps one and two of evaluation is quite long. Because of the use of selective search, the majority of the time is spent generating proposals. For this reason, in order to be able to recognize and detect both with high accuracy and faster, the Faster R-CNN model is used in step three. During the evaluation of the models after the training process, it is observed that traffic signs are detected and recognized faster with higher accuracy with the Faster R-CNN. When the total test time for all three steps are compared, the detection and recognition time for the models are approximately 51, 63, and 7 seconds, respectively. Although the Faster R-CNN model provides high accuracy in a short test time, it is still considered slow to be used in real time. Future work includes running the developed Turkish TSR system on a network server, detecting and recognizing the traffic signs in real time. Also, it also includes improving the success values of the models for more classes by collecting more data.

REFERENCES

1. Turkish Statistical Institute. Table-3 Number of road motor vehicles 2021 [cited 2021 27 November]. Available from: https://data.tuik.gov.tr/Bulten/Index?p=Motorlu-Kara-Tasitlari-Ekim-2021-37432.

2. Bucsuházy K, Matuchová E, Zůvala R, Moravcová P, Kostíková M, Mikulec R. Human factors contributing to the road traffic accident occurrence. *Transportation Research Procedia*. 2020;45: 555–561.

3. Turkish Statistical Institute. Table-6 Faults causing road traffic accidents involving death or injury in Turkey, 2020 [cited 2021 27 November]. Available from: https://data.tuik.gov.tr/Bulten/Index?p=Road-Traffic-Accident-Statistics-2020-37436.

4. Stallkamp J, Schlipsing M, Salmen J, Igel C. The German traffic sign recognition benchmark: a multi-class classification competition. *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2011. p. 1453-1460.

5. Houben S, Stallkamp J, Salmen J, Schlipsing M, Igel C. Detection of traffic signs in real-world images: the German traffic sign detection benchmark. *Proceedings of International Joint Conference on Neural Networks*. IEEE Computer Society Press. 2013. p. 715-722.

6. Ballard DH, Brown CM. *Computer vision*. Englewood Cliffs, N.J: Prentice-Hall; 1982.

7. Hubel DH, Wiesel TN. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology.* 1959;148(3): 574–591.

8. Papert SA. The summer vision project. 1966 [cited 2021 17 March]. Available from: https://dspace.mit.edu/handle/1721.1/6125.

9. Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks.* 1988;1(2): 119–130.

10. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE.* 1998;86(11): 2278–2324.

11. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision.* 2015;115(3): 211–252.

12. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. *Communications of the ACM.* 2017;60(6): 84–90.

13. Maldonado-Bascon S, Lafuente-Arroyo S, Gil-Jimenez P, Gomez-Moreno H, Lopez-Ferreras F. Road-sign detection and recognition based on support vector machines. *IEEE Transactions on Intelligent Transportation Systems.* 2007;8(2): 264–278.

14. Garcia-Garrido MA, Sotelo MA, Martin-Gorostiza E. Fast traffic sign detection and recognition under changing lighting conditions. *IEEE Intelligent Transportation Systems Conference.* IEEE; 2006. p. 811–816.

15. Barnes N, Zelinsky A, Fletcher LS. Real-time speed sign detection using the radial symmetry detector. *IEEE Transactions on Intelligent Transportation Systems.* 2008;9(2): 322–332.

16. Albawi S, Mohammed TA, Al-Zawi S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET).* Antalya: IEEE; 2017. p. 1–6.

17. Jung S, Lee U, Jung J, Shim DH. Real-time Traffic Sign Recognition system with deep convolutional neural network. In: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI).* Xian, China: IEEE; 2016. p. 31–34.

18. Chaudhari T, Wale A, Joshi A, Sawant S. Traffic sign recognition using small-scale convolutional neural network. *SSRN Electronic Journal.* 2020.

19. Novak B, Ilic V, Pavkovic B. YOLOv3 Algorithm with additional convolutional neural network trained for traffic sign recognition. In: *2020 Zooming Innovation in*

Consumer Technologies Conference (ZINC). Novi Sad, Serbia: IEEE; 2020. p. 165–168.

20. Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH, USA: IEEE; 2014. p. 580–587.

21. He K, Zhang X, Ren S, Sun J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2015;37(9): 1904–1916.

22. Girshick R. Fast r-cnn. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE; 2015. p. 1440–1448.

23. Boujemaa KS, Berrada I, Bouhoute A, Boubouh K. Traffic sign recognition using convolutional neural networks. In: *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. Rabat: IEEE; 2017. p. 1–6.

24. Qian R, Liu Q, Yue Y, Coenen F, Zhang B. Road surface traffic sign detection with hybrid region proposal and fast R-CNN. In: *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. Changsha, China: IEEE; 2016. p. 555–559.

25. Ren S, He K, Girshick R, Sun J. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017;39(6): 1137–1149.

26. Zhang Z, Zhou X, Chan S, Chen S, Liu H. Faster r-cnn for small traffic sign detection. In: Yang J, Hu Q, Cheng M-M, Wang L, Liu Q, Bai X, et al. (eds.) *Computer Vision*. Singapore: Springer Singapore; 2017. p. 155–165.

27. Yang T, Long X, Sangaiah AK, Zheng Z, Tong C. Deep detection network for real-life traffic sign in vehicular networks. *Computer Networks*. 2018;136: 95–104.

28. DiCarlo JJ, Zoccolan D, Rust NC. How does the brain solve visual object recognition? *Neuron*. 2012;73(3): 415–434.

29. Yamashita R, Nishio M, Do RKG, Togashi K. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging.* 2018;9(4): 611–629.

30. CS231n: Convolutional Neural Networks for Visual Recognition Stanford - Spring 2021 [cited 2021 29 March]. Available from: https://cs231n.github.io/classification/.

31. Wang S-C. Artificial neural network. In: *Interdisciplinary Computing in Java Programming.* Boston, MA: Springer US; 2003. p. 81–100.

32. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics.* 1943;5(4): 115–133.

33. Single-Layer Neural Networks and Gradient Descent by Sebastian Raschka in Mar 24, 2015; [cited 2021 31 March]. Available from: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

34. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review.* 1958;65(6): 386–408.

35. Ivakhnenko AG, Lapa VG. Cybernetic Predicting Devices. *CCM Information Corporation.* 1965.

36. Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks.* 1988;1(2): 119–130.

37. Khan S, Rahmani H, Shah SAA, Bennamoun M. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision.* 2018;8(1): 1–207.

38. Phung VH, Rhee EJ. A deep learning approach for classification of cloud image patches on small datasets. *Journal of information and communication convergence engineering.* 2018;16(3): 173–178.

39. Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine.* 2012;29(6): 141–142.

40. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556 [cs]*. 2015.

41. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861 [cs]*. 2017.

42. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C. Mobilenetv2: inverted residuals and linear bottlenecks. *arXiv:1801.04381 [cs]*. 2019.

43. Uijlings JRR, van de Sande KEA, Gevers T, Smeulders AWM. Selective search for object recognition. *International Journal of Computer Vision.* 2013;104(2): 154–171.

44. Felzenszwalb PF, Girshick RB, McAllester D, Ramanan D. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2010;32(9): 1627–1645.

45. J. Hale. Deep learning framework power scores 2018 [cited 2021 4 July]. Available from: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a.

46. G. B. Team. Tensorflow 2015 [cited 2021 5 July]. Available from: http://tensorflow.org.

47. Chollet F. Keras 2015 [cited 2021 5 July]. Available from: https://keras.io.

48. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. *Nature.* 2020;585(7825): 357–362.

49. Hunter JD. Matplotlib: a 2d graphics environment. *Computing in Science & Engineering.* 2007;9(3): 90–95.

50. OpenCV (Open Source Computer Vision Library) [cited 2021 9 November]. Available from: https://opencv.org/about.

51. Scikit-learn 1.0.2 [cited 2021 12 November]. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

52. Xmltodict 0.12.0 [cited 2021 13 November]. Available from: https://pypi.org/project/xmltodict/.

53. Bisong E. Google colaboratory. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform.* Berkeley, CA: Apress; 2019. p. 59–64.

54. TensorFlow 1 Detection Model Zoo 2018 [cited 2021 30 August]. Available from: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md.

55. COCO (Common Objects in Context) 2015 [cited 2021 29 August]. Available from: https://cocodataset.org/#detection-eval.