Road Sign Identification

HL Potgieter

30057698

School of Electrical, Electronic and Computer Engineering

Mini dissertation submitted in partial fulfilment for the requirements of the degree *Bachelor of Engineering* in Computer and Electronic Engineering

Supervisor: Prof W. Venter
Subject: EERI474 Project
24 October 2022

ABSTRACT

The project allocated to the student is identified and the customer's needs are expressed. The problem is identified as an algorithm to detect road signs under different conditions. A hypothesis is made about the expected solution to the problem. The requirements for the problem are defined, and the student's deliverables are stated. The student will develop a web-based application to identify and classify different road signs. The existing solutions to the problem are also discussed, and the different methods of the algorithms are described.

Different solutions to solve the problem is discussed in Chapter 2 which includes making use of Deep learning methods to recognize the road signs. The different methods are compared in Chapter 3 to make an informed decision on which solution to use. Other components that will be used for the project are also chosen.

Chapter 4 and Chapter 5 deals with the implementation of the solution. The YOLOv5 network was chosen for the detections. A custom database was created to include road signs found in South Africa. The database consists of signs from the Mapillary dataset and the GTSRB dataset. The YOLOv5m6 and YOLOv5l6 models were trained and tested against each other to see which performed better. The 5l6 had higher accuracy detections at lower computation speeds whereas the 5m6 model had more detections with lower accuracy at a higher computation speed. Chapter 6 deals with the conclusions as well as the recommended further work.

Key words:

Computer vision, Deep learning, Road sign identification, YOLOv5

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Abbreviation	Definition
AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
B	Blue
CIE	Commission Internationale de l'Éclairage
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CYMK	Cyan, Magenta, Yellow and Key
DOTA	Dataset of Object deTection in Aerial images
F1	Precision and recall metrics
FPS	Frames Per Second
G	Green
GPU	Graphics Processing Unit
GTSRB	German Traffic Sign Recognition Benchmark
HSV	Hue, Saturation, Value
HTML	HyperText Markup Language
mAP	Mean Average Precision
NA	Not Applicable/ Not Available
R	Red
RGB	Red, Green, Blue
ROI	Region Of Interest
RSI	Road Sign Identification
SSD	Single Shot Multibox Detection
SVM	Support Vector Machine
VGG	Visual Geometric Group
VOC	Visual Object Classes
XML	eXtensible Markup Language
YCbCr	Green (Y), Blue (Cb), Red (Cr) (digital video color space)
YIQ	Y stands for luminance part and IQ stands for chrominance part
YOLO	You Only Look Once
YOLOV5	You Only Look Once Version 5

CHAPTER 1 INTRODUCTION

1.1 Purpose of the document

This report documents the context and the objectives of the project assigned to the student. It provides background information on the project as well as a hypothesis on how it will be solved. It also includes the technical requirements of the project.

1.2 Background

Humans have always attempted to find new and easier ways to do tasks. The computer was introduced into everyday life during the third industrial revolution. The machine's utility developed quickly, and new applications for these devices were discovered. One application is to replace humans in order to perform activities more accurately, efficiently, and continuously. In 1979, the first computer program with the purpose of recognition was created and its function was to distinguish between different handwritten numerals [1]. Computers have evolved significantly since then to get to where we are now. We were introduced to modern self-driving cars utilizing computers in 1977 when Japan started to use cameras and a computer to do the processing [2]. In 1995, the first self-driving car was tested on public roads.

Since then, other methods of detecting things have been utilized and improved upon. There has also been an increase in computing power. More efficient algorithms could be implemented to analyse data faster and distinguish more between various objects. To design the next generation of a road sign identification (RSI) algorithm, it is necessary to take advantage of the resources available and to expand on the most recent studies of object recognition.

1.3 Problem Statement

A web-based application should be used to identify and classify road signs from images and videos captured in various lighting conditions.

1.4 Hypothesis

An algorithm will be used with the capability of detecting different types of road signs. It will be developed as a web application.

1.5 Project Objective

1.5.1 Primary objective

The algorithm should be able to distinguish between the various signs based on their shape and colour. The sign should be easily identifiable from several angles and at various times of the day. The algorithm should also be capable of detecting signs with additional variations, such as colours and shapes similar to those of signs.

1.6 Anticipated Benefits of solution

This algorithm would be a significant step forward in computer vision. It could be used to replace human drivers in autonomous cars, lowering the dangers of travel caused by human mistakes. Poor vision, inclement weather, weariness, and distractions are examples of these errors. It increases the level of safety for passengers, cars, and other road users. It may take the role of monotonous occupations like long-distance transport, where numerous human drivers have fallen asleep at the wheel and caused tragic accidents, or it could take the place of taxi services, making it safer for passengers who ride with strangers. By adjusting the formulas to what the algorithm is looking for, the algorithm's core can be used for different computer vision tasks.

1.7 Technical Requirements

1.7.1 Requirements

- The algorithm will identify different road signs.
- The algorithm will distinguish between different colours consisting of red, blue, and green.
- The algorithm will distinguish between different shapes consisting of angular, round, hexagonal and rectangular.
- The algorithm will detect signs under different lighting conditions.
- The algorithm will detect signs from different angles.
- The algorithm will make a prediction as close to real time as possible.
- The algorithm will be upgradeable with new features.
- The software will be web-based.
- The user will get feedback as to whether a sign was identified and the type of the sign.
- The user will get feedback on the information displayed on the sign.

1.7.2 Scope Definition

The student will be responsible for the research and development of the algorithm. This includes gathering sufficient data for the project as well as the front-end and back-end programming of the software. This also includes the testing of the algorithm with different cases, such as edge cases where different signs are present in a single image, or the sign is barely visible due to heavy rain. The student will not be responsible for the integration of the algorithm into an external project.

1.8 Deliverables

The student will deliver a functioning web-based algorithm capable of identifying road signs in different conditions and in different environments. The algorithm will comply with the requirements as stated under the heading 1.7.1 Requirements.

1.9 Conclusion

Since the invention of the computer, computer vision has come a long way. With the help of recent studies and modern hardware, the student will take it upon himself to develop an algorithm to detect and identify road signs. The project needs to adhere to the requirements stipulated in the document.

CHAPTER 2 LITERATURE SURVEY

2 Introduction

One of the earliest recorded cases where computer vision was implemented is during the summer vision project [3]. It was an attempt to use summer workers effectively through the construction of a significant part of a visual system. The project was chosen to encourage independent and individual work by segmenting the problem into sub-sections while still being a part of the complex problem of pattern recognition. The goal of the project was to construct a system of programs that will divide a vidisector picture into regions of: likely objects, likely background areas and chaos [3]. Shape and surface properties were the main components that needed to be analysed for the first goal. The second goal was to describe the region and tied in with the first goal. The final goal was object identification which would name the objects from a known object vocabulary.

In the 1970's Kunhiko Fukushima started research on neural networks with multiple pooling and convolutional layers [1]. In 1979 Kunhiko designed the first artificial multi-layered neural network capable of learning through mimicking the brain's visual network [4]. He named this first neural network "Neocognitron." This network was capable of flexible pattern recognition and machine learning. This was done by changing the learning rules to modify interactions amongst the cells and the internal hidden layers [4]. This networked started the deep neural network research which is the cornerstone of visual pattern recognition.

In the 1980's backpropagation and feed-forward paths were introduced for Neural networks. Both of these concepts form part of modern Deep learning [1]. Backpropagation was then used in 1989 by Yann LeCun to "read" handwritten numbers. Artificial neural networks can work through more data and detect patterns that humans cannot recognize. These types of algorithms can be found in financial institutes trading stocks or credit scoring, being used against crime and terrorism, in social media apps to filter content and to promote custom content to individual according to preference [5]

2.1 Feature extraction from Images and Videos

An image is a two-dimensional signal where the light intensity at any pixel is a function of two spatial variables. When working with videos the signal becomes a three-dimensional signal where the third dimension is time [6]. A colour image consists of three 2-D signals that represents the colours red, green, and blue. A colour video signal is a 3-channel signal composed of three 3D signals. This can also be changed for different colour spaces where the 3-channel signal can consist of a luminance component and 2 chrominance components [6].

A colour can be represented by a set of numbers and a colour model is the mathematical model of that set [7].The different colour spaces that can be found is: RGB, HSV, CIElab, YIQ, YCbCr, and CYMK. These different colour spaces have different properties and work differently in different conditions [8]. Different components of colour spaces can also be utilized together to increase the chance of recognition [8]. As seen in Figures 2.2 and 2.3, the colour of a road sign is reflected differently depending on the time of day as well as environmental conditions.

Figure 2.1 Warning sign image taken in the night

Figure 2.2 Stop sign image taken on a bright day

HSV is shown as the best performing colour space for all environmental conditions including nighttime and rain [8]. As standard images are taken in the RGB colour space. To convert from RGB to HSV colour space the following equations is used for each individual pixel [9].

$$H(Hue) = 60^{\circ} \times H' \tag{1}$$

$$H' = \begin{cases} \dfrac{G-B}{C} mod 6, & if\ M = R \\ \dfrac{B-R}{C} + 2, & if\ M = G \\ \dfrac{R-G}{C} + 4, & if\ M = B \\ undifined\ if\ C = 0 \end{cases} \tag{2}$$

$$C = M - m \tag{3}$$

$$M = \max(R, G, B) \tag{4}$$

$$m = \min(R, G, B) \tag{5}$$

Where:

R, G, B is the red, green, and blue values of the pixel.

C: Saturation

M: value

Through the use of thresholding of the primary colour of the sign, can the sign be extracted from the image by converting the image to a binary image where the colour in the image matches the threshold as seen in Figure 2.4. A binary image is an image that is quantised to two values 1 and zero where 0 is black and 1 or 255 is white. When this is applied to an image using a threshold the resulting picture will be black except for the parts with colour in the threshold which will be changed to white. This can be observed in Figure 2.3. This also creates noise which is effectively pixels with the same colour as in the threshold that was changed to white. By filtering the noise from the resulting image, the shape of the sign can be clearly seen. The pixels which form a solid shape is classified as the Region of Interest. The ROI can then be extracted from the image by taking the pixel coordinates and extracting the coordinates from the HSV image. What is left is an extracted traffic sign or noise if it was a false ROI. This is seen in Figure 2.4. This enables the recognition algorithm to only look at a portion of an image instead of a whole image. The sign can also be outlined with the use of bounding boxes [10].

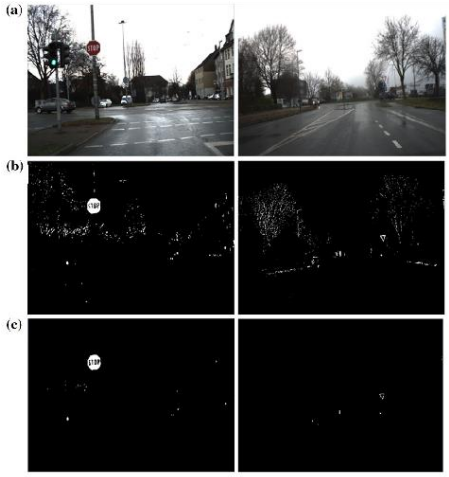


Figure 2.3 a) Image captured, b) Binary image, c) Binary image with noise removed

Figure 2.4 Region of Interest Identified extracted traffic signs

Also, important to consider the size and file type of the image/video as well as the positioning of the camera. Where applicable portions of the image can be cut out if a common obstruction can be found throughout the image series. Image augmentation can be used to expand the set of training images. This is done by tweaking the properties of each image such as the colour, cropping, and brightness. This also reduce the model's dependence on certain factors of an object [10].

2.2 Machine learning and Deep Learning

The simulation of the human intelligence process by machines is known as artificial intelligence. AI systems work by processing data, analyzing it, and detecting patterns to make predictions about future states [11]. Artificial intelligence can be divided into six branches of, Machine Learning, Deep Learning, Natural Language Processing, Robotics, Expert Systems, and Fuzzy Logic. This paper will only focus on Machine learning and Deep Learning as the other branches are not applicable to this problem.

Machine learning can be defined as the application of AI where algorithms process data, learn from it and make informed decisions by using the knowledge obtained. Deep learning can be defined as a subfield of machine learning that utilizes layer structured algorithms that form artificial neural networks that is capable of self-learning and making intelligent decisions [12]. Deep learning is a subset of machine learning, but deep learning has the advantage of human like decision making.

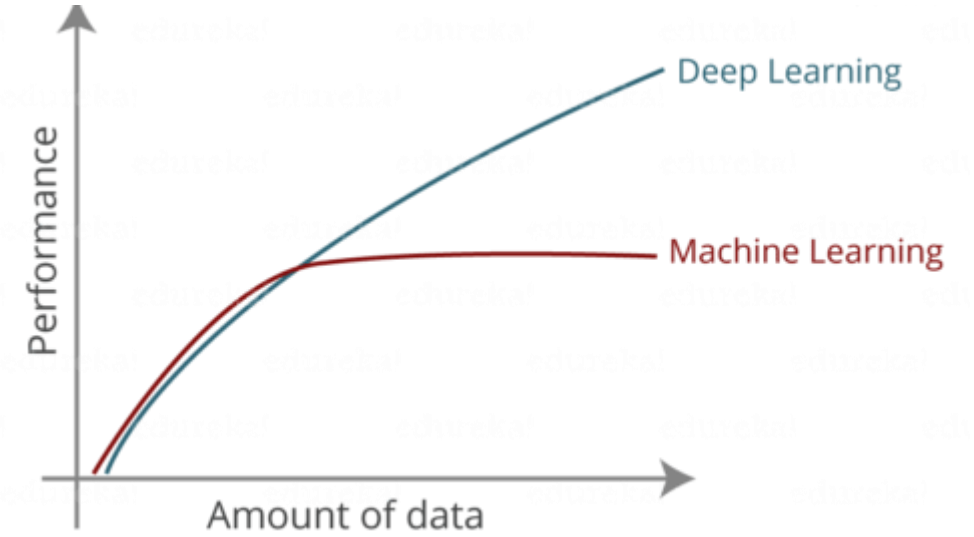


Figure 2.5 Machine Learning compared to deep learning regarding amount of data processed and performance

Machine learning consists of three different types of learning methods namely Supervised-, unsupervised- and semi supervised learning. Supervised learning is when the network is given a dataset containing the features as well as the labels and tasked with producing a model to predict the label given the input feature. A feature-label pair is called an example. The model is provided with labeled examples that the supervisor has chosen with specific parameters. The supervisor can thus specify the exact outcome the model should produce for a specific input. This type of learning is used for applications such as text to speech, machine translation or object detection [10]. Unsupervised learning consists of finding interesting transformations of input data without any help from targets. The model is given data and must draw its own conclusion to the relevance between different inputs. This type of learning is more used for understanding new data before using it in supervised learning or finding correlations between different inputs. [13]

To function Machine learning needs three things: Input data points, examples of expected output and a way to measure the algorithm is doing the correct job [13]. The central problem in machine learning is to learn useful representations of the input data. "Learning" is the process of

automatically searching for better representations [13]. In deep learning the layered representations are learned via neural networks which is shown in Figure 2.6

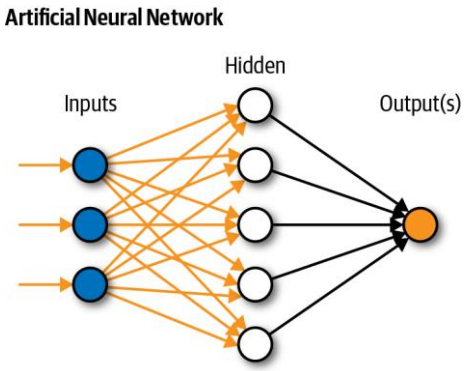


Figure 2.6 Representation of a neural network

A Neural Network consist of three layers. The input layer, the hidden layer, and the output layer. All these layers consist of nodes connected to the preceding and the succeeding layer. The layers can be any size and the hidden layer can consist of multiple layers Each node consists of a weight, and it is the specification of what the layer does to the input data. The weights of the nodes are changed during the training phase to get an output node to be activated for a certain input and have a specific outcome This is done with the following equation:

$$y = \sum (f \times w_x) + b$$

Where:

f: activation function

x_x : input

w_x: weight

b: bias

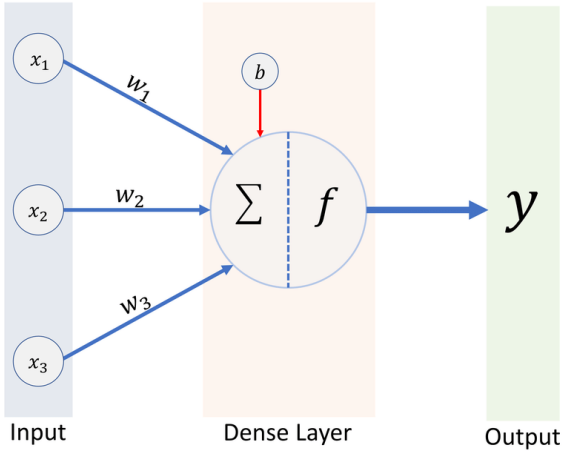


Figure 2.7 Neural network equation calculation

An activation function like a sigmoid function can be applied on the calculated node to apply a certain activation characteristic and decrease sensitivity on activations. The loss function is used to measure the distance of the output from the expected output. This is done by taking the

predictions of the network and the target outcome and computing a distance score. This score is then used as feedback by the optimizer to adjust the weights of the layers to reduce the score for the current example. When created, the layers are all assigned random weights and is far from the desired outcome, but with each example the weights are adjusted closer to the desired outcome. A trained network is one where the loss function score is at a minimal [13].

To successfully train a Deep learning model for computer vision we need to extract the data hidden in the image and find patterns within it. We will assume that there can be multiple objects in a single image, that we need the position and the category of. Bounding boxes are rectangular boxes that specifies the x- and y- coordinates of the object in the image. This can be defined in the (x, y) coordinates of the four corners or the (x, y) coordinates of the center and the width and height of the box [10]. Anchor boxes is used to sample large regions in the input image and to determine if a region contains an object of interest. Anchor boxes is multi bounding boxes with varying scale and aspect ratio centered on each pixel. Anchor boxes is displayed in Figure 2.8.

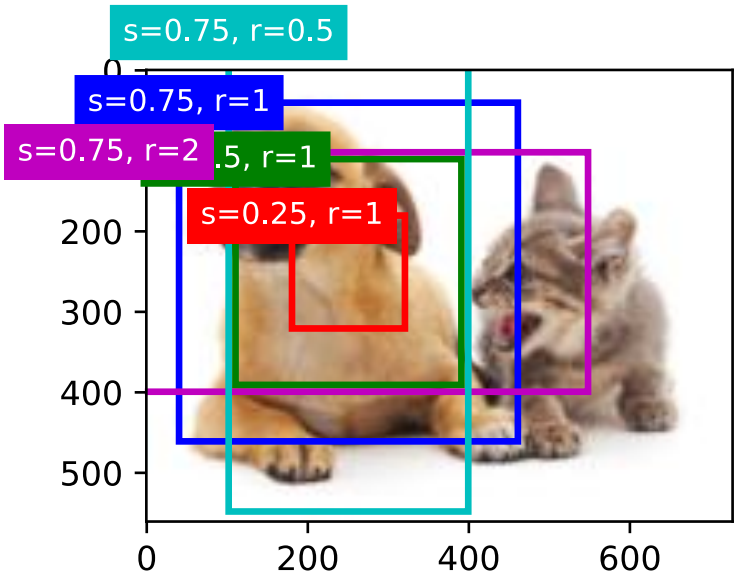


Figure 2.8 Image showcasing multiple bounding boxes

These boxes represent samples of the image. Multiscale anchor boxes limit the amount of anchor boxes needed by only using a few pixels to center around. Different scales can be used for larger and smaller object detection in the images [10]. These methods are performed in different types of algorithms that will be discussed in Chapter 2.3.

2.3 Different Networks

A Support Vector Machine (SVM) is part of classification algorithms that is known as kernel methods. SVM's is used to find a best fit decision boundary between two different data boundaries. This can be seen in Figure 2.9. The boundary is found by mapping the data to a high dimension representation and expressing the boundary as a hyperplane, and maximizing the margin by computing a good boundary through maximizing the distance between the hyperplane and the closest data points from each class which enables the boundary to generalize new samples outside the training set [13].

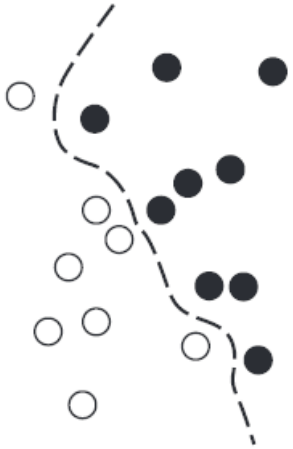


Figure 2.9 Decision boundary between two data sets

Max pooling's role is to down sample feature maps aggressively. This operation consists of the extraction of windows from the input feature map an outputting the max value of each channel. This operation is like that of a convolution, whereas a convolution transforms local patches via a learned linear transformation, they are transformed with a hardcoded max tensor operation. This is done to induce special-filter hierarchies that force successive convolution layers to look at increasing large windows, as well as to reduce the number of feature map coefficients [13].

R-CNN

Region based Convolution Neural Networks extracts many region proposals (normally 2000) from an input image [14] Feature vectors is then computed for each proposed region using a CNN and then classifies each region using class specific linear SVMs. The region proposals are generated using selective search to enable controlled comparison with prior detection work. The feature extraction is done with a CNN. The region is converted to be used as input to the CNN, which entails a resolution change done with wrapping [14]. The extracted feature is then outputted for the region proposal through forward propagation. The SVMs is trained to classify objects by taking the extracted features as well as the labelled class of each region as input. Figure 2.10 and Figure 2.11 shows the workings of the R-CNN algorithm.

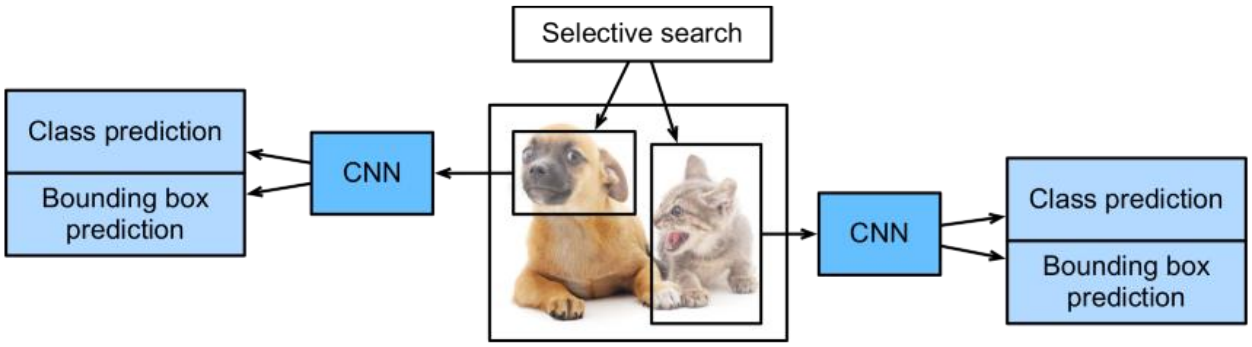


Figure 2.10 The R-CNN model

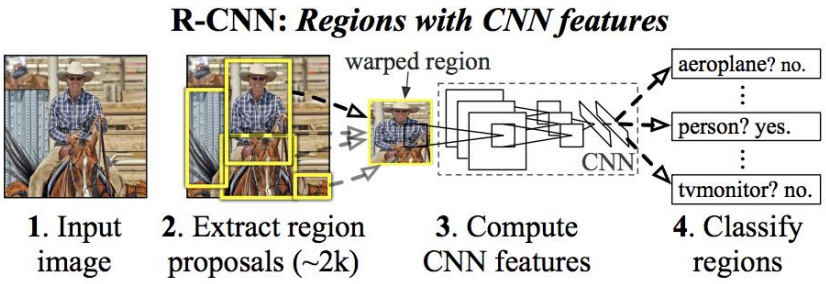


Figure 2.11 Explanation of R-CNN model

Fast R-CNN

The R-CNN is slow although it uses pretrained CNN. This is due to the large amount of region proposals and the CNN having to check every region for features. The bottleneck is the independent CNN forward propagation for each region without the sharing of computation. [15] Computations are repeated due to overlaps in the proposed regions. The solution to this problem is the Fast R-CNN algorithm. The Fast R-CNN only performs CNN forward propagation on the whole image instead of each proposed region. [10] The Fast R-CNN takes as input an image and a set of object proposals. The shape of the CNN can be defined as:

$$1 \times c \times h_1 \times w_1 \quad (6)$$

Where $h_1 \times w_1$ is the convolution kernel and c is the number of channels for data input.

The CNN then produces a convolution feature map through max pooling and several convolutions. From the convolution feature map, we can identify regions of proposals and warp them into squares using the ROI pooling layer that concatenate features of shapes that are extracted further for all proposed regions. The concatenated shape is described as:

$$n \times c \times h_2 \times w_2 \quad (7)$$

Where n is the number of

The fully connected layer then transforms the concatenated features into an output of n × d where d is dependent on the model design. The class and bounding boxes are then predicted. The class prediction uses SoftMax regression [10]. (SoftMax regression is the generalization of logistic regression that is used for multi-class classification). This algorithm has the advantage of doing one convolution on the entire image and then creating a feature map instead of doing a convolution for each ROI. Figure 2.12 Shows the different layers of the Fast R-CNN

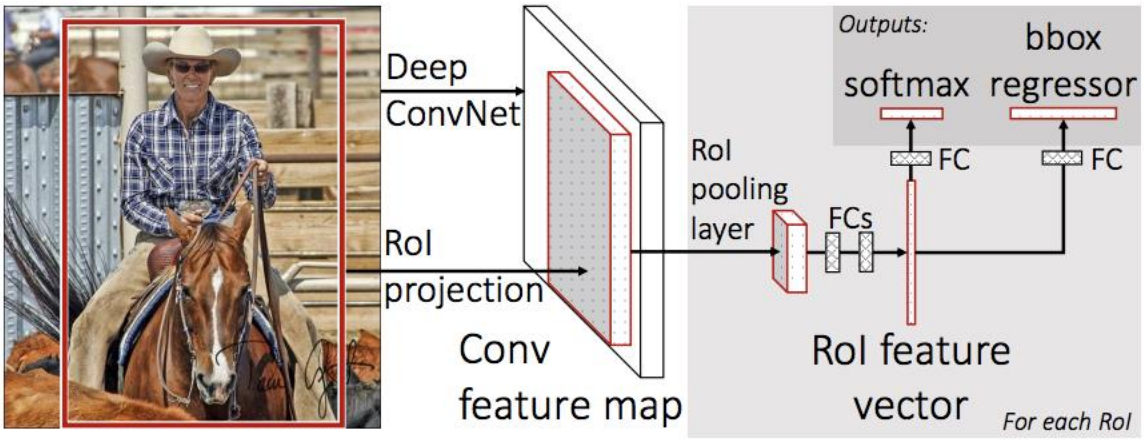


Figure 2.12 Fast R-CNN workings

Faster R-CNN

To increase the accuracy of the Fast R-CNN network more region proposals must be generated in selective search. To combat the loss of accuracy by reducing the region proposals, a region proposal network is introduced to the Faster R-CNN replacing the selective search of the Fast R-CNN. The predicted region proposals are reshaped by using a RoI pooling layer. The pooling layer then classifies the image within the proposed region, as well as predict the offset values of the bounding boxes [10]. The proposals network is jointly trained with the rest of the model. This means that the proposal network learns how to generate high-quality region proposals. Figure 2.13 shows the different layers of the Faster R-CNN.

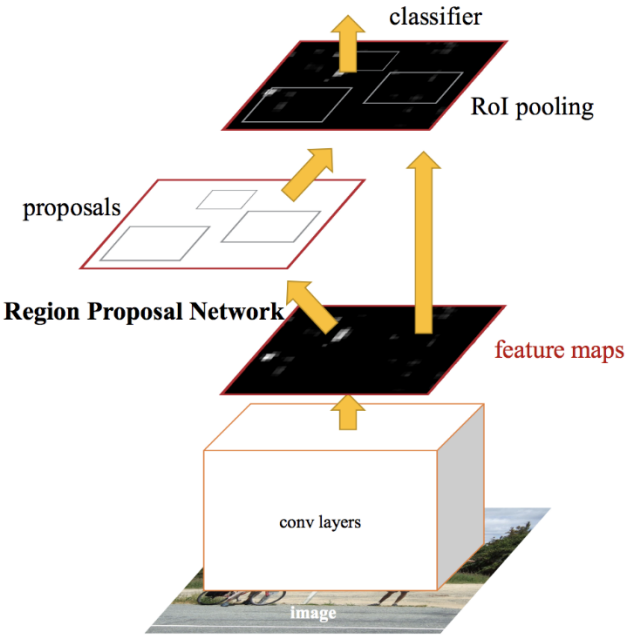


Figure 2.13 Faster R-CNN workings

YOLO – You Only Look Once

YOLO works very different from the previous mentioned networks. These networks do not look at the whole image but instead uses regions that localize the object in the image. YOLO utilizes a

single CNN to predict the bounding boxes and the class probabilities for these boxes. It does this by splitting an image into a n×n grid and within each grid there are m bounding boxes. If the bounding boxes have a class probability above a certain threshold, it is selected and used to locate the object in the image [16]. A representation of this process is seen in Figure 2.14

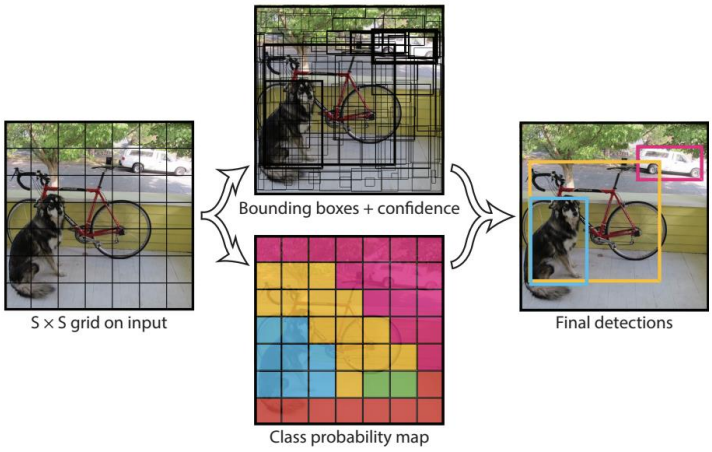


Figure 2.14 YOLO algorithm representation

YOLO was originally created by Joseph Redmond in 2016. Since then, he released YOLOv2 [16] in 2017 and YOLOv3 [17] in 2018. YOLOv2 was implemented to improve upon detecting small objects in groups and the localization accuracy. This was done by introducing batch normalization, which in turn increases the mAP value (mAP is the mean Average Precision). YOLOv2 enabled multi bounding boxes from a single cell improving from the single class/cell of YOLOv1. YOLOv3 was created to improve YOLO with the latest CNNs that used residual networks and skip connections. YOLOv3's can detect features at 3 different scales, and in combination with the three bounding boxes per cell at each scale, enables this network to have nine anchor boxes. This further improved the accuracy of small object detection and detection speed [18]. In April 2020 Alexey Bochkovskiy published YOLOv4 [19] which was an improvement on the previous version. YOLov4 introduced a lot of architectural improvements from YOLOv3 although it is not seen as an official YOLO algorithm. YOLOv5 was released on 18 May 2020 shortly after YOLOv4 and uses the Pytorch framework. It is maintained by Ultrlaytics and is open source.

Single Shot Multibox Detection (SSD)

SSD was also created along with YOLO to be a replacement for R-CNN. It was first introduced in 2016 by Wei Liu [20]. The detector works by generating varying anchor boxes with different sizes and detects varying-size objects in these boxes by predicting classes and offsets of the anchor boxes. This is done with a base network and multiple multiscale feature map blocks, in a single forward pass of the network. The VGG-16 forms the base network, due to its robust performance. The size of the input of each layer is decreased by using auxiliary convolutional layers instead of the original VGG fully connected layers. This allows for extraction of different scale objects. Figure 2.15 shows the VGG architecture.

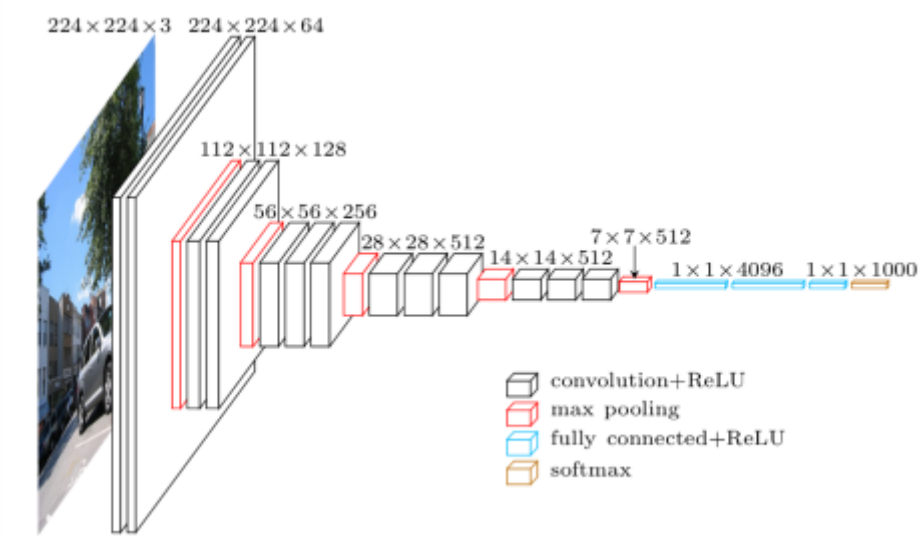


Figure 2.15 VGG Architecture for SSD

Szegedy developed the MultiBox method for fast class-agnostic bounding boxes. This allowed the network to also calculate the confidence loss (Measure of objectness of computed bounding box) and the Location loss (Measure of prediction distance from ground truth bounding box). Priors or anchors (fixed sized bounding boxes that closely match the distribution of the original ground truth boxes) is pre-computed to have an Intersection over Union (IoU) of > 0.5. The priors are then used to regress closer to the ground truth boxes. Figure 2.16 explains IoU.

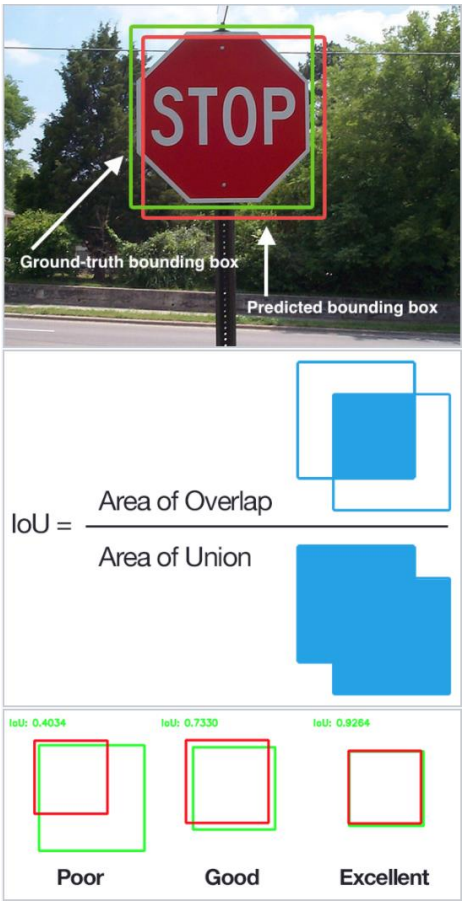


Figure 2.16 Diagram of IoU explanation

This speeds up the process as the network starts from the prior's location and is close to the ground truth box than starting from a random location. The prior selection also increases the generalization of the network due to the priors being used have a IoU above 0.5, without the need for a pre training phase.

2.4 Dataset manipulation

The dataset used should be split into three groups: training- validation and test sets. The training data is the largest portion of the data and is used to train the model. The validation set is a small percentage of the data that incorporates every different instance of the dataset. This is used to validate the model once it is trained. The model should be changed using the validation data, but it should not be overfitted to it as this will compromise the model. The model will then perform well as it was made for the validation data but will fail when other data is used. To prevent this hyperparameters should only be changed once for each parameter to limit the data leak into the model of the validation data. The test data is also a small percentage of the data that incorporates the different instances of the dataset as well as edge cases. The test data is used for demonstration purposes only and once the model cannot have seen this data before the presentation [13].

To get the best model result the data needs to have a few characteristics:

- The data must be relevant. This means that the data should be relevant to the problem and should be real-world data such as that that will be processed by the algorithm. This is due to the network only capable of making an informed prediction on data similar to that which it was trained on.
- The data should be properly classified. This entails that the data should be labeled properly for the classes that the model will be trained on. This includes the use of bounding boxes to make the training process more accurate.
- The data should be correctly formatted as the network accepts a fixed size input. If the image sizes differ it should be resized to a standard size.
- The dataset needs to be substantial. About 25000 instances of a class is required to train an accurate model. Different techniques can be used to fill a dataset that lacks instances such as data augmentation. Other solutions are training methods such as K-fold where the training and validation sets is changed for every batch to make it seem that the dataset is "larger" than it actually is.
- The dataset needs to be diverse to showcase different instances that the model will encounter to prepare it for scenarios where edge cases is produced, and to produce accurate results on inaccurate inputs.

2.5 Conclusion

This chapter discussed the main components needed for the project as well as the different techniques that have been used. Each major component was investigated, and different solutions were proposed. In Chapter 3 these solutions will be compared to aid in the decision making of the final solution for the problem.

CHAPTER 3 DESIGN

3 Introduction

This chapter deals with the design process of a website that is capable of road sign identification. The problem will be broken into different pieces that will be looked at. The different solutions as discussed in Chapter 2 will be compared with each other and then a solution will be chosen. This chapter also highlights the student's engineering contribution to this project.

3.1 Design Problem Definition

The problem can be defined as:

- Input images should be collected of traffic signs from different angles, have different lighting conditions, be of the working environment where the system will be used, and be plentiful.
- The images should be sorted as well as formatted to be accessed by the algorithm.
- An algorithm should be used which is accurate, can function in real time and be able to generalize.
- The algorithm should be trained and validated for the best accuracy.
- The application should be web based and its output should be of the input image with a bounding box on the identified road sign with the road sign name as well as the confidence level.

A functional block diagram of the project design is shown in Figure 3.1.

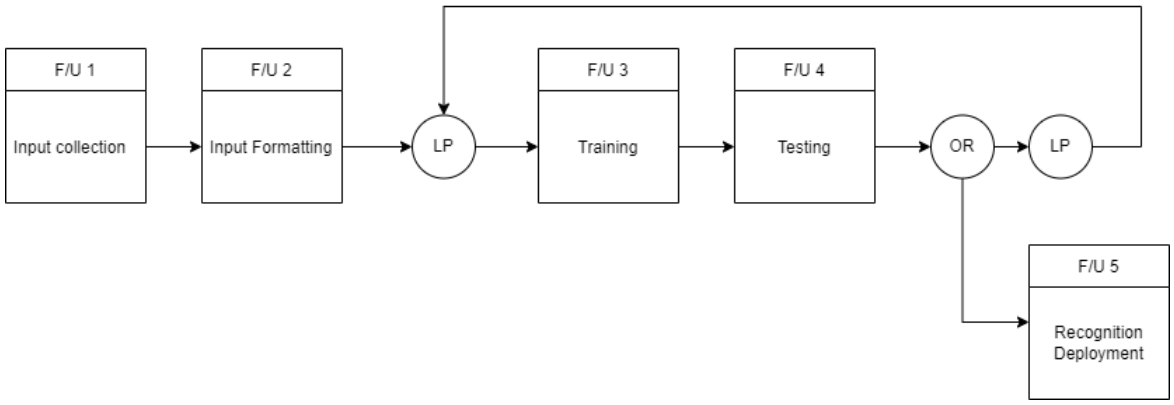


Figure 3.1 Diagram of system operations

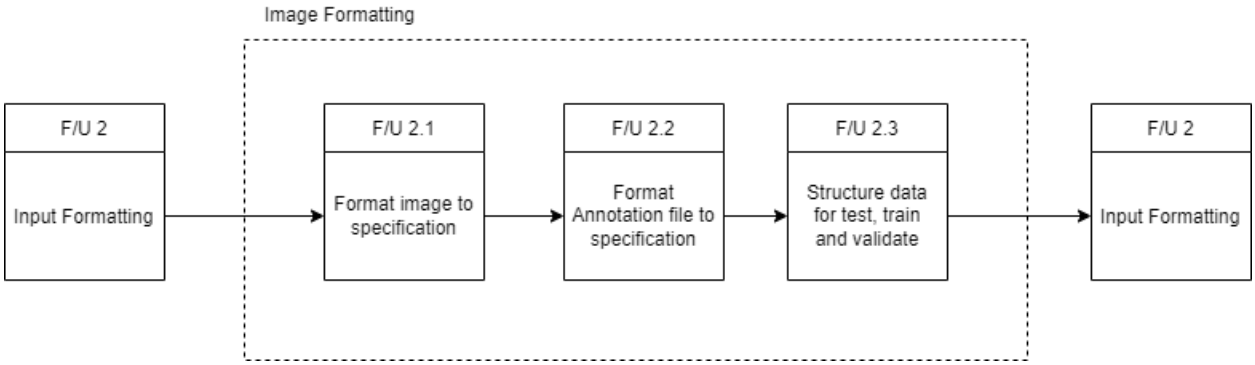


Figure 3.2 Diagram of the Input Formatting

Figure 3.2 displays the workings of the Input formatting operation. The collected dataset should be studied, and the relevant formatting of the images should be done such as scale changes, input filtering and annotation file conversion. The data should then be stored in a structured format to be used by the algorithm.

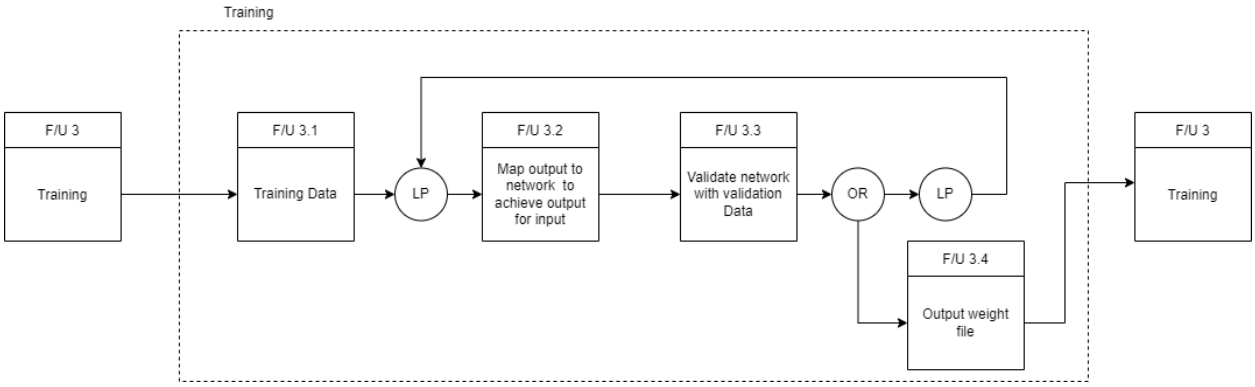


Figure 3.3 Diagram of the Training procedure

Figure 3.3 shows the training procedure of the model. Training data is taken and fed into the model. The weights are then adjusted to fit the output. The process is repeated for every image in the training set. After each run the model is validated with the validation set to check the accuracy of the model. This process is repeated for the number of loops. The weights file is then outputted to be used in the model. This file sets the node values to be used as the trained model. This model is in the web application and its input is images that is loaded onto the web page.

3.2 Component selections

Different components of the project need to be evaluated and compared to make an informed decision on which to choose to move forward with.

3.2.1 Dataset

Different methods exist to acquire data, whereas 3 of the most common ways is to get a dataset that someone has already compiled, search the internet for images related to the problem and building your own dataset or lastly to use a camera and collect data to be used. The dataset needs to meet a few requirements, and the best method to obtain the data which meet these requirements will be chosen.

- The database needs to have a substantial number of entries for each class.
- The data needs to be as close to the application environment as possible.
- The images need to be labelled and have bounding boxes assigned to the relevant portion of the image.
- The images need to be of road signs found in South Africa

Seeing that road sign identification is a new application to machine learning means that there does not exist many databases that consist of enormous amounts of labelled data. Databases with 5000+ entries for each class that is labelled does not exist for the public or have not been compiled. To create a database with that scale would take a lot of man hours to collect the data and to label it as each image needs an accompanying file with the bounding box location and the class id for the bounding box. It can be done whether through driving around and taking pictures or through the use of images on the internet, the amount of time it will take to label a database that size will mean that a team of people is needed to do it, or an algorithm to do it which comes back to the purpose of the project.

A solution to this problem is using data from multiple datasets. This is less time consuming than collecting the data manually as the data is already labelled. The data entries should be converted to the same format and stored in the same train, test, and validation folders.

Mapillary is a company that uses user collected street imagery to update and create street maps of various kinds. The images are created by users that uses dashcams on their vehicles and the imagery is uploaded to the website where it is incorporated into an updated street view map. Mapillary have been using Machine learning to update maps with unique features such as showing the location of road signs on a map. They have created a database that consist of more than 100 000 high resolution images with 300 different classes of signs to be used by anyone for Road sign identification purposes. The images are taken from a dashcam that is in the exact environment where the project will be used. The images are taken in different times of day in different weather conditions and from different viewpoints on a vehicle. The drawback of the database is that the images is from around the world and different signs is used in different countries but there are no more than 5 different versions of each type of sign and commonly used signs in South Africa is included. (https://www.mapillary.com/dataset/trafficsign).

Figure 3.4 Image from Mapillary Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) is a common dataset used in RSI applications. The dataset was created for the International Joint Conference on Neural Networks

in 2011. The dataset consists of more than 50 000 images over 40 classes. The dataset consists of cropped images of road signs that is depicted in various locations, different weather conditions and angles. Although there is not a large variety of different images, each image is augmented with different scales and different resolutions. Even though it is of German traffic signs some of the traffic signs still apply on South African roads. These signs should be extracted from the dataset and can be used in conjunction with other datasets to create a larger dataset. (https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign)

Figure 3.5 Image from GTSRB dataset

Combining these two datasets will result in 150 000 theoretical images. This amount will decrease as signs that is not used in South Africa is removed from the dataset, but the remaining images should be enough to train a model with a high accuracy. The combination of the two datasets will enable the network to generalize and to detect signs in different environmental conditions from the viewpoint of a vehicle. The exact signs that will be used will be influenced by the number of images there is in the combined dataset.

3.2.2 Algorithm Selection

To compare the different type of algorithms we need to look at a few different aspects on which to compare them. We will look at:

- The accuracy of the network (mAP and F1 scores [21])
- The Frames per Second (FPS) to measure the speed
- The robustness of the algorithm.

The different algorithms have been discussed in Chapter 2.3 and will not be discussed again. The different algorithms that will be compared is Faster R-CNN, SSD YOLOv3, YOLOv4 and YOLOv5. YOLOv4 and YOLOv5 is fairly new and haven't been tested rigorously, as such YOLOv3 have been the go-to for RSI applications [22] [23]. Testing each of these models and comparing them form the dataset will consume a lot of time, instead previous comparisons done in the last few years will be used to make an informed decision on which algorithm to use for this project.

Comparisons

The different algorithms are compared in different studies with different data. The results are displayed in Table 3.1.

Table 3.1 Results of the comparisons between Faster R-CNN, SSD and YOLO

Reference	Dataset used	Algorithms	Results
[24]	Remote sensing satellite images Train: 826 Test: 275 Resolution: 300 - 1000	Faster R-CNN YOLOv3 SSD	YOLOv3 had a higher mAP and FPS than the other algorithms
[25]	Google Earth And DOTA Train: 224 Test: 56 Resolution 600 - 1500	Faster R-CNN SSD YOLOv3	YOLOv3 had a higher mAP and FPS than the other algorithms
[26]	Korean Express Dataset Train: Test: Resolution: NA	Faster R-CNN SSD YOLOv4	YOLOv4 had a higher accuracy (mAP) SSD had a higher FPS

There are not many studies where these comparisons are performed and in some cases the versions of the algorithms used is not specified making the data useless. Looking at the results in Table 3.1 we can see that overall, the YOLO algorithms perform better. SSD does outperform YOLO in some cases regarding FPS and Accuracy it have been reported to have a lot more false positives. This means that a detection is made even though there is no object present. Faster R-CNN have been reported to have a higher accuracy than YOLO and SSD but due to the low FPS it cannot be used in a real time environment.

Seeing that the YOLO algorithm performs better overall, the different versions will be compared next, and the results will be displayed in Table 3.2.

Table 3.2 Comparisons between YOLOv3, -v4 and -v5

Reference	Dataset used	Algorithms	Results
[27]	MS COCO Train: 118 000 Test: 5000 Resolution: NA	YOLOv3 YOLOv4 YOLOv5	YOLOv5 has a higher mAP than -v3 and -v5l YOLOv3 has a higher FPS than -v4 and -v5
[28]	DOTA aerial image Train: NA Test: NA Resolution: NA	YOLOv3 YOLOv4 YOLOv5l	YOLOv5l has the highest mAP YOLOv3 has the highest FPS
[29]	Blood Cell Count dataset (Shenggan) Train: 874	YOLOv3 YOLOv4 YOLOv5	YOLOv3 takes much less time to train

	Test: NA Resolution: 416		YOLOv5 and v4 had better accuracy YOLOv4 was chosen for the lower training time
[30]	Sign Language Dataset Train: 2000 Test: NA Resolution: NA	YOLOv3 YOLOv4 YOLOv5	YOLOv5 had the highest accuracy YOLOv3 had the highest learning rate

Looking at the results in Table 3.2 we can see that overall YOLOV5 had the highest accuracy of the three versions and that YOLOv3 had the highest learning rate. YOLOv4 had the highest accuracy in some tests with some classes but overall, it came in second. All the tests were done without changing the hyperparameters to give the networks a fair test. By changing the hyperparameters the network can learn faster as well as become more accurate. YOLOv5 was chosen to be used for the RSI. The YOLO algorithm is more accurate and faster in real time than SSD and Faster R-CNN and comparing the YOLO versions with each other v5 was overall the top contender. With its high accuracy and high FPS, it can be incorporated into a self-driving vehicle in the future.

3.2.3 Programming Languages

The language that the project needs to be developed in needs to be selected. Due to the project being a web-based solution it needs a frontend and backend programming language. The frontend is the part that the user interacts with, and the backend is the part that does the execution of the task.

Backend

YOLOv5 was created using the Pytorch framework. Pytorch is an open-source machine learning framework based on the Python programming language. It is used for rapid prototyping and deployment. For this reason, it will be easier to use Python as backend for the project. Python is a programming language that has become more popular due to the available libraries for machine learning and web development. There are ample web application frameworks and Python holds its own against other programming languages and JavaScript environments such as Node.js. Django and Flask will be used for the backend development.

The model will be trained on a local machine seeing that a server won't have the computational capacity to perform the training. As such the training application will be done in Python. The image formatting will also be done in python as Python has a large collection of libraries with functions that makes image processing and file handling easier.

Frontend

The frontend is the part of the application that the user is in contact with. This user experience is the major concern with this section as a badly designed user interface can jeopardise the usage of the project. JavaScript and HTML will be used to create a web page that is easy to use as well as aesthetically pleasing. Bootstrap will be used in this regard as it consist of a variety of customizable pre-built components that is user friendly.

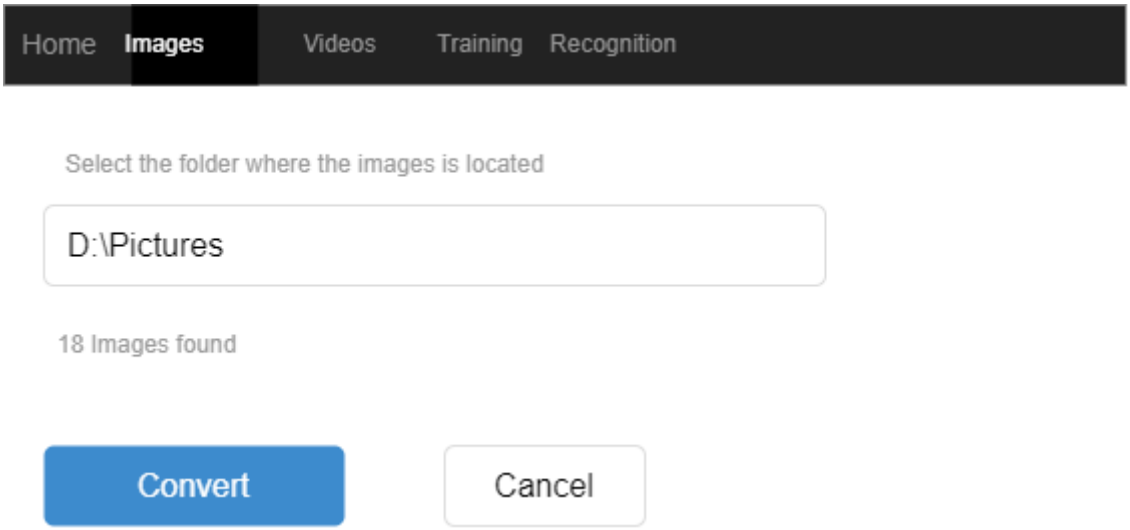


Figure 3.6 Wireframe of the web application

Figure 3.6 displays a wireframe of the layout of the web application. The site will consist of a home page, recognition page and a reference page. The home page will give the user a brief overview of how the application works and how to use it. The reference page will consist of all relevant information and supporting documents needed to understand the working of the application and how it was achieved. The recognition page will consist of a place to upload an image or video and to process it through the algorithm to detect a road sign. The resulting image or video will then be displayed. Pre-processed images and videos can also be displayed on the page.

CHAPTER 4 EXPERIMENTAL DESIGN

4.1 Introduction

This chapter deals with the detail design of the RSI project. The chapter will look at the different aspects which forms the final solution. The proposed solution in Chapter 3 will be implemented. The data collection, handling and extraction will be discussed. The Training setup as well as the performance metrics to measure the success of the network will be discussed in this chapter.

4.2 Data and Environment

4.2.1 Data Collection

In Chapter 3 it was decided to combine 2 datasets for the most images as there doesn't exist a South African Road Sign dataset. The first is the Mapillary dataset containing about 100 000 images of 300 different classes. The existing problem with this dataset is that it contains images of road signs from all over the world. These unwanted signs should be removed as they will cause the network to be inaccurate and the aim of this project is to recognize South African road signs. The other dataset that will be used is the GTSRB that consist of 50 000 German traffic sign images of 40 different classes. Most of the German traffic signs is the same as South African traffic signs but not all. These signs that is not found in South Africa should be removed. Both datasets consist of different sized images ranging from 28 × 28 pixels to 5660 × 2830 panorama images.

Every image in the dataset is accompanied with a file containing information on the location of the road signs in the image. For the Mapillary dataset this is in the form of a json file. For the GTSRB dataset all the image's information is in a single csv file. An example of the files is shown in Appendix A. The Mapillary database consist of two different annotated datasets. The one part is annotated by humans consisting of 52 000 of the images and the other 48 000 images is computer generated annotations. The annotations are the information on the bounding boxes stored in the accompanying file. This contains the class of the sign, the position of the bounding box around the sign on the image and the dimensions of the image. The file is connected to the relevant image by sharing the same name in the case of the Mapillary database and containing the image name in the annotation in the case of the GTSRB database. An example of a bounding box from the annotation drawn on the image is displayed in Figure 4.1.

Figure 4.1 Bounding Box drawn on image

4.2.2 Data details and Handling

Both datasets were divided into training and validation images. The human annotated files of the Mapillary dataset also contained a test set. The training set is used for the training of the network. The validation set is used to validate the network accuracy to improve the network, and the test set is used for demonstration purposes only. To sort the dataset and remove the unwanted signs, a Python script was written to check which classes of signs were available. The algorithm would go through each of the json files from the Mapillary dataset and if a new class is detected it would add it to a list. Some of the json files contains multiple sign information as multiple signs of relevant classes are present in the image. This list was then written to a text file to be sorted through and to remove the classes of unwanted signs. The Python script used for this purpose is displayed in Appendix B.

During this process it became apparent that there exist images in the test and validation sets that does not have annotation data in the json file. The json file would be empty. This caused problems as the script would crash at the empty file. To manually remove the empty files along with the images corresponding to the file would be impossible. Another Python script was created to check if the json file was not empty and if there was a corresponding image. If the conditions were not met the filename was written to a text file and the corresponding files were removed. The algorithm used is displayed in Appendix B. The text file was used in the same extraction algorithm to count the different classes of signs.

With the sign classes of the Mapillary dataset extracted the unwanted classes could be removed. The Mapillary API was used to check if the sign is used in South Africa. The Mapillary API is an interactive webpage that lets users upload Streetview data taken from dashcams in vehicles. The images are displayed as breadcrumbs on the map. As the Mapillary API make use of their own RSI system it displays the images taken of road signs and displays them. To check if a sign exists in South Africa the name is used in the filter and if an image of a sign is detected on the map the class is saved to a text file. This file then contains all the classes of street signs that is available in South Africa.

The unwanted signs could not be removed from the database as some of the images contain multiple signs and not all the signs in the image are unwanted. This is due to some overlap in road signs in countries for instance the octagonal stop signs are universally used in most countries.

The GTSRB dataset is more structured than the Mapillary dataset. The GTSRB is divided into folders for each class. It also has a folder showing an image for each of the classes. The classes that corresponded to the Mapillary dataset were kept while the other were removed. The unwanted classes can be completely removed since the GTSRB's images are all closeups of the signs. Of the classes only 29 were selected that is present in South Africa. The complete list is available in Appendix B.

To ensure a more accurate and faster network the number of classes needs to be reduced. This will be done by satisfying the requirement of as many as possible training images. It was decided to add all the computer annotated images to the training set as well as all the GTSRB images to the training set. This is done instead of splitting the sets into 80% training and 10% validation and 10% testing. The human annotated set has a large enough validation and testing set since the testing set does not have annotation files. To reduce the classes another Script is needed to count all the images for each class to determine which classes have the most instances and will be used. The algorithm that was used is a modified version of the algorithm used to extract the

annotations for each image and is displayed in Appendix C. The workings of the algorithm will be discussed in more detail in Chapter 4.2.3.

The algorithm worked by opening every json file and checking if the class is present in the used classes. This is done for each sign present in the json file. In the case of the GTSRB database the csv file was traversed, and each instance of a class being used was counted. The instances of classes for both the Mapillary and GTSRB databases were counted together. From the 29 classes only 12 had more than 900 instances, excluding the “other-sign” class. This class was excluded because it showed every sign that were not assigned a class. This meant that a mix of shapes and colours made up this class. To prevent inaccuracy and false detections on the images it was decided to discard this class. The instances do not represent the number of images for each class but rather the number of signs of that class present in all the images.

Table 4.1 displays the number of instances for the training and the validation as the test images cannot be counted due to not having annotation files.

Table 4.1 Classes with amount of each extracted from the databases

Class	Training instances	Validation instances
information--parking--g1	2098	201
regulatory--height-limit--g1	976	58
regulatory--maximum-speed-limit-40--g1	1618	124
regulatory--maximum-speed-limit-60--g1	2411	533
regulatory--no-entry--g1	3785	615
regulatory--no-heavy-goods-vehicles--g1	1285	196
regulatory--one-way-straight--g1	2170	455
regulatory--one-way-left--g1	1039	170
regulatory--stop--g1	3129	444
regulatory--yield--g1	6827	1067
warning--road-bump--g1	1323	53
warning--slippery-road-surface--g1	1163	183

Figure 4.2 displays each of the signs that will be used. The signs are in the order as presented in Table 4.1, going from left to right, up to down.

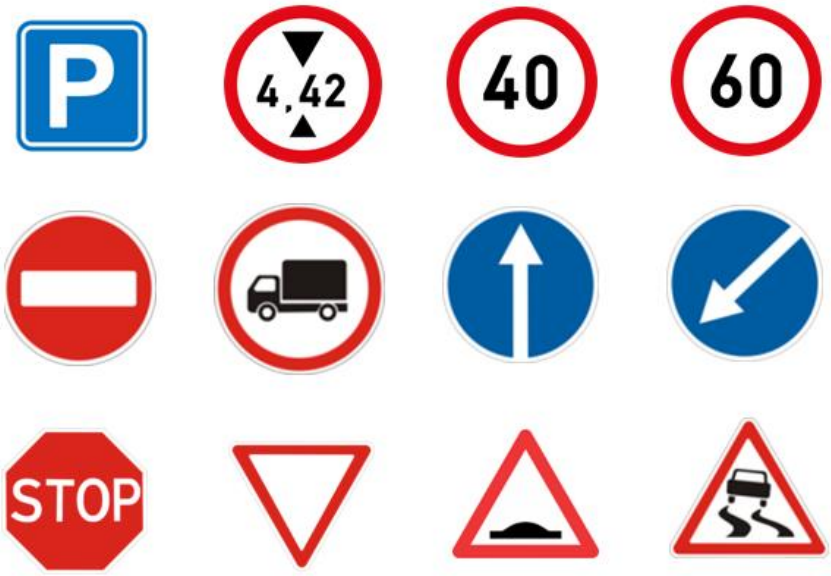


Figure 4.2 Images of the classes that will be used

The final classes that will be used were written to a text file to be used as input in the algorithm which will order the images as well as write the correct annotation file for each image. This will be discussed in Chapter 4.2.3.

4.2.3 Annotations and Dataset Creation

Before training the Yolov5 network, the dataset needs to be conditioned. This can be done as the classes that will be used have been identified. The YOLO algorithm requires a specific dataset structure to be able to function. Firstly, the dataset needs to be in the folder structure as shown in Figure 4.3. This means that all the images need to be in the same folder with the train, validation, and test sets in their respective subfolder, the same goes for the labels.

Figure 4.3 Folder structure of YOLO format

The dataset also needs to include a data.yaml file. This file contains the path to each of the sets, the number of classes and the list of the classes present in the dataset. The file used is shown in Appendix C.

Annotations

YOLO requires the annotations in a specific format. The annotations must be a txt file and the bounding box details must be in the YOLO format. The YOLO format is shown in Figure 4.4 below. All the elements are separated with a space. Every line represents a bounding box on the image.

```
1 0.408 0.30266666666666664 0.104 0.15733333333333333
1 0.245 0.424 0.046 0.08
```

Figure 4.4 Example of annotation file

The first element is the class number. This number corresponds to the position, of the class in the bounding box, which is in the list present in the yaml file shown in Appendix C. The next element is the x coordinate of the centre of the bounding box. The third element is the y coordinate of the centre of the bounding box. The fourth element is the width of the bounding box, and the last element is the height of the bounding box. If Figure 4.4 is analysed, we can see that the image has two bounding boxes of the second class in the list. The last requirement is that the annotation file must have the same name as the image.

The Mapillary and GTSRB annotations are in the wrong format and needs to be changed to be used in the YOLO algorithm. The Mapillary annotations is in the form Pascal VOC XML. This format can be seen in Appendix A. It provides the bounding box coordinates as a set of four x- and y coordinates of the corners of the bounding box. It also provides the class label of the bounding box. The file also contains the width and heigh of the image. The GTSRB dataset contains annotations in the csv format. A cell contains the width and height of the image, the x and y coordinates of the bounding box corners, the class id, and the path to the image.

To convert these formats to the YOLO format we will use the following equations.

$$x = (xmin + xmax)/2 \quad (8)$$

$$y = (ymin + ymax)/2 \quad (9)$$

xmin and xmax is the x coordinates of the bounding box width.

X is the x coordinate of the centre of the bounding box.

ymin and ymax is the y coordinates of the bounding box height

Y is the y coordinate of the centre of the bounding box.

$$w = ymax - ymin \quad (10)$$

$$h = ymax - ymin \quad (11)$$

w is the width of the bounding box.

h is the height of the bounding box.

$$dw = 1/width \quad (12)$$

$$dh = 1/height \quad (13)$$

dw is the reciprocal of the width of the image.

dh is the reciprocal of the height of the image.

$$x = x * dw \quad (14)$$

$$y = y * dw \quad (15)$$

$$w = w * dw \quad (16)$$

$$h = h * dw \quad (17)$$

Each of the calculated bounding box properties were calculated with respect to the height and width of the image. The same formula can be applied to the GTSRB annotations.

To convert both database's annotations to the YOLO format the algorithm in Appendix C was used. The algorithm was written to incorporate both the Mapillary and GTSRB databases. For the Mapillary database the annotation files are in the same folder while the images are separated into testing, validation, and training. The annotation files were traversed, and the file name were checked in the three image folders to see if the image exist. If the image does not exist the file is deleted, else the folder is recorded as the corresponding set (train, validation and, testing). For each file, the bounding box information is extracted. If the class is not in the desired list, then the class id is changed to "other-sign." If the file consists of only "other-sign" classes, then the file is deleted along with the image. If the file contains one or more of the desired classes, then the "other-sign" classes are removed from the list and the remaining bounding box information is converted to YOLO format according to Equation 8 – 17. The file is saved in the txt format with a number as its name each set of image and annotation files receive a unique name. Both files are placed in the relevant folder according to the set it is a part of.

The same process was done for the GTSRB with the difference being the folder structure of the database. And the unwanted classes being removed prior due to the images of each class contained in their respective folder.

The final database contained 12 classes, 1000 testing images, 23 000 training images and 3650 validation images. The test image set contains a mix of correct classes as well as images without any correct classes. This will be left in to test the network against false positives, to see if any detections are made on the image.

4.3 Training the network

4.3.1 Training setup

Google Colab was used for the training due to the high-performance hardware available. Colab is a collection of virtual machines ranging in hardware specifications. This can cause problems as the resources is assigned to each user randomly and according to their usage intensity. This is also affected by the users demand and by the user's subscription level. In the end Google Colab pro+ was used as it provides a background execution option letting the operation run while the browser is closed for up to 24h. Getting assigned higher performance hardware is also more common resulting in a quicker training time. This makes the project difficult to reproduce as different hardware was used for training sessions and the parameters such as the epoch and batch sizes were adapted to be used on the specific hardware assigned to the session.

The database was uploaded to Google Drive to provide easy access to it for training. YOLOv5 was downloaded using the GitHub repository. Two different loggers were used to keep track of the training. The first was ClearML and the second was Weights & Biases(W&B). The loggers keep track of the whole training procedure. The network parameters are logged to recreate the experiment. The loggers also keep track of the performance metrics and creates graphs to review the performance of the training session. W&B also saves the weights of the network after every epoch is completed which enables the training process to be resumed if the training failed.

4.3.2 Model selection

The different models that are available in the YOLOv5 network is shown in Figure 4.5. The models are compared on their speed and their average precision using the COCO dataset.

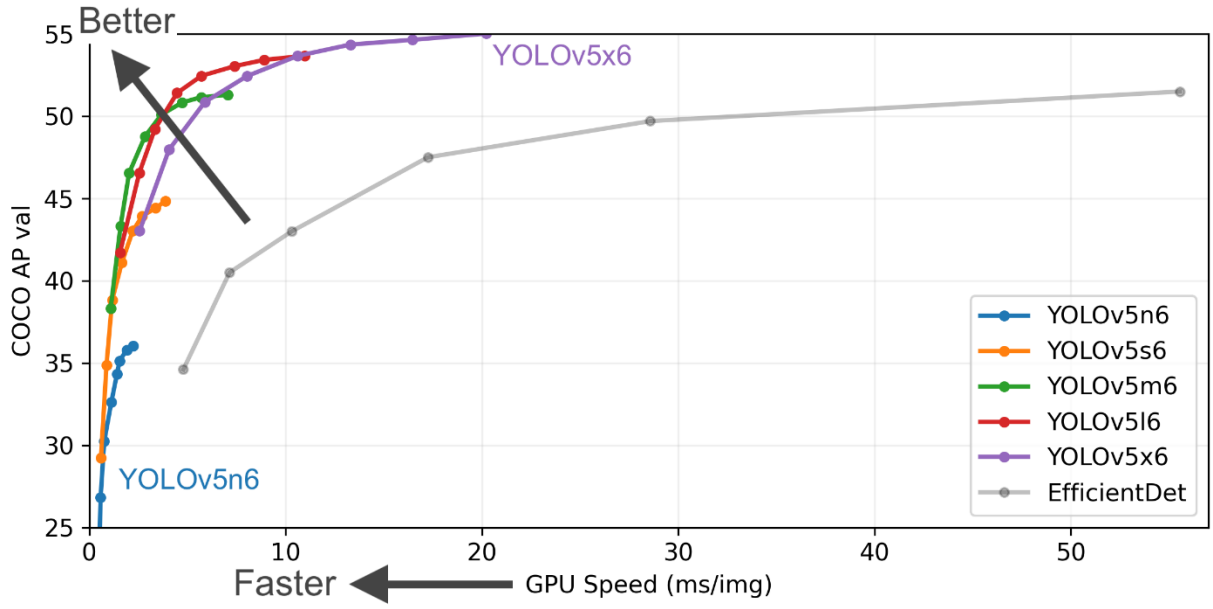


Figure 4.5 YOLOv5 models detection speed vs Average precision

The models that will be used are the YOLOv5m6 and the YOLOv5l6. Both models show a fast detection time with a high average precision. Both models allow for a larger image input at 1280 pixels where the YOLOv5m and YOLOv5l only allow for a maximum of 640-pixel input. The image size makes up the first layer in the network. The larger pixel count is chosen as small details in the images can be missed if the resolution of the image is smaller than the image itself. The larger input layer will cause the detection speed to go down, but the accuracy will not be compromised. The YOLOv5m6 network architecture is shown in Figure 4.6. The YOLOv5m6 model will also be tested with a smaller image size to see if the accuracy is compromised and if a faster detection time is recorded.

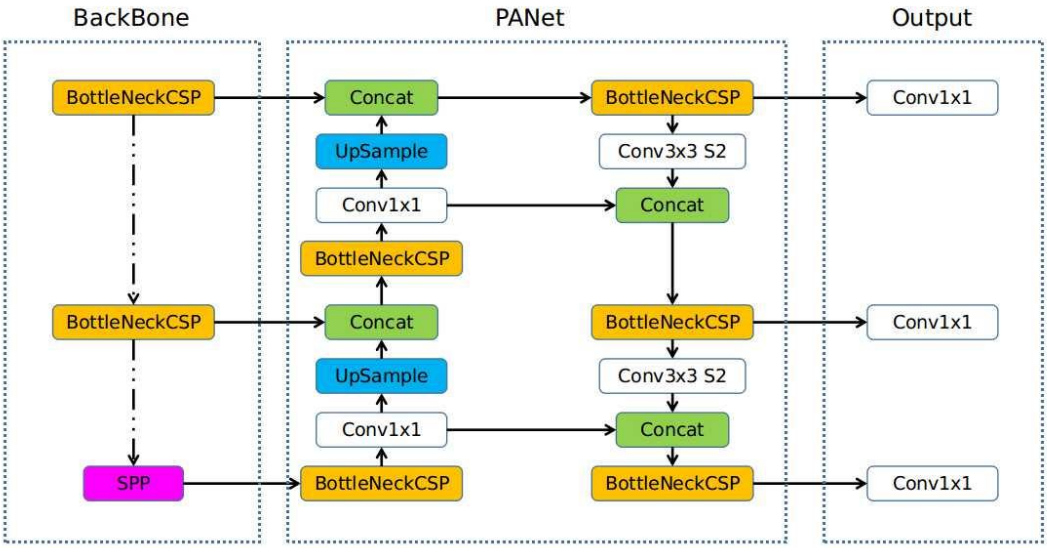


Figure 4.6 YOLOv5 model architecture

The specific architectures of both models are available in Appendix D. The summary of the two models is shown in Table 4.2.

Table 4.2 Summary of YOLO models architecture

Model	Image size	Layers	Parameters
YOLOv5m6	320	276	35312412
YOLOv5m6	1280	276	35312412
YOLOv5l6	1280	477	76247116

It can be observed that the YOLOv5l6 model is the largest of the models with the most layers and parameters.

4.3.3 Performance indicators

The network needs to be measured in a way to see how well it is performing. The metrics that will be used are listed in Table 4.3.

Table 4.3 Descriptions of performance metrics for network

Metric	Description
mAP 0.5	Mean Average Precision for > 0.5
mAP 0.5:0.95	Mean Average Precision for different IoU thresholds from > 0.5 to > 0.95.
Precision	Ration between the Positive classifications and the total classifications.
Recall	Ratio between the positive correct classifications and the total positive samples.
Class loss	The correctness of the classification.

Box loss	How close the dimensions of the prediction bounding box is to the ground truth box.
Object loss	Correct object classified.

The Intersection over Union is shown in Equation 18.

$$\mathrm{IoU}(\mathrm{A},\mathrm{B}) = \frac{\mathrm{A} \cap \mathrm{B}}{\mathrm{A} \cup \mathrm{B}} \tag{18}$$

The mAP can then be described as:

$$mAP = \frac{1}{|\mathbf{classes}|}\sum_{\mathbf{c}\in\mathbf{classes}} \frac{\#\mathbf{TP}(\mathbf{c})}{\#\mathbf{TP}(\mathbf{c}) + \#\mathbf{FP}(\mathbf{c})} \tag{19}$$

Where $TP_{(c)}$ is the True positive and $FP_{(c)}$ is the False Positive.

The average precision for the class c is:

$$\frac{\#\mathrm{TP}(\mathrm{c})}{\#\mathrm{TP}(\mathrm{c}) + \#\mathrm{FP}(\mathrm{c})} \tag{20}$$

The mAP and the precision need to be 1 to be a perfectly trained model while the object-, class- and box loss needs to be 0. This means that the exact same bounding box is placed on the exact same spot with the correct class for all the validation cases. It also means that no false positive detections were made. Combined, these metrics will show how well the network can detect signs in an image and if the prediction is correct.

CHAPTER 5 IMPLEMENTATION AND EVALUATION

5.1 Introduction

This chapter deals with the training of the models and the evaluation of the trained models against each other and the specified performance requirements.

5.2 Results from training runs

The training runs were conducted as described in Chapter 4. After every run the best model's weight is taken and used as the weights for the next model. This ensures that the network builds upon the previous run. The runs for the YOLOv5m6 model are shown below in the Table 5.1, and the results of those runs are also displayed in Table 5.2. The run along with the performance metrics of the best performing epoch are displayed. The combined epoch results of all the runs are available in Appendix E.

Table 5.1 YOLOv5m6 training runs

Run	Experiment ID	Link
1.1	45b50c35a6104bdb8a30fbd9ccf34954	https://app.clear.ml/projects/bec26561a1ee4dfd84253db58747f678/experiments/45b50c35a6104bdb8a30fbd9ccf34954/output/execution
1.2	Devoted-brook-25	https://wandb.ai/projek/YOLOv5/29diqkox
1.3	Wandering-darkness-5	https://wandb.ai/projek/YOLOv5/2ox3bp96
1.4	Stilted-puddle-9	https://wandb.ai/projek/YOLOv5/2mq4gknj
1.5	Wobbly-fog-38	https://wandb.ai/ projek/YOLOv5/208rplsr

Table 5.2 YOLOv5m6 training runs performance metrics

Run	Run weights used	mAP 0.5	mAP 0.5:0.95	Precision	Recall	Box loss	Class Loss	Object Loss
1.1	-	0.5811	0.4048	0.5934	0.5818	0.0338	0.0211	0.0097
1.2	1.1	0.6111	0.5947	0.6079	0.5947	0.0281	0.0173	0.0084
1.3	1.2	0.7459	0.57588	0.7728	0.7192	0.0243	0.0086	0.0072
1.4	1.3	0.7845	0.5891	0.7825	0.7438	0.0097	0.0045	0.0035
1.5	1.4	0.7693	0.5917	0.7573	0.7325	0.0092	0.0039	0.0045

The same was done for the larger YOLOv5l6. And the results are shown in Table 5.3 and the performance metrics in Table 5.4.

Table 5.3 YOLOv5l6 training runs

Run	Experiment ID	Link
2.1	Sparkling-oath-13	https://wandb.ai/projek/YOLOv5/2sen733u
2.2	Fanciful-monkey-36	https://wandb.ai/projek/YOLOv5/mor1wjsb
2.3	Prime-lion-37	https://wandb.ai/projek/YOLOv5/2c7f2nea

Table 5.4 YOLOv5l6 training runs performance metrics

Run	Run weights used	mAP 0.5	mAP 0.5:0.95	Precision	Recall	Box loss	Class Loss	Object Loss
2.1	-	0.7668	0.5833	0.7691	0.7252	0.0092	0.0043	0.0044
2.2	2.1	0.7845	0.6037	0.7913	0.7466	0.0089	0.0038	0.0041
2.3	2.1	0.7782	0.5940	0.7558	0.7506	0.0088	0.0045	0.0042

Lastly the YOLOv5m 320-pixel model is trained to see if it performs better than the larger models. The results of this training session are displayed in Table 5.5 and the performance metrics in Table 5.6.

Table 5.5 YOLOv5m training runs

Run	Experiment ID	Link
3.1	Resilient-salad-42	https://wandb.ai/projek/YOLOv5/grusxgey

Table 5.6 YOLOv5m training runs performance metrics

Run	Run weights used	mAP 0.5	mAP 0.5:0.95	Precision	Recall	Box loss	Class Loss	Object Loss
3.1	-	0.7668	0.5833	0.7691	0.7252	0.0092	0.0043	0.0044

Observing the performance metrics of the different models we can clearly see that the 5l6 and 5m6 models perform the best. The 5m model didn't perform as well due to the down scaling of the images resulting in signs with resolutions of less than 3 pixels. The 5m model will be discarded from this point forward.

5.3 Detection speed and confidence

The different trained models were tested on their detection speed and accuracy by taking the best model of each and using images that the networks haven't seen as the benchmark. The

detection speed for each model using the same runtime in Google Colab is shown in Table 5.7. The GPU that was used for the detections was a Nvidia A100 SXM4 40GB

Table 5.7 Best training run for each model

Model	Best model
YOLOv5m6	Stilted-puddle-9
YOLOv5l6	Fanciful-monkey-36

The chosen testing images shows the different environments that the system needs to work in. It deals with signs that is barely visible. There are images taken on a very bright day and other where the sun is setting. All these variables influence the output. The images that will be looked at are available in Appendix F.

Table 5.8 shows the detection speeds and detections of the 5m6 model and the 5l6 model for a confidence of 0.5. The confidence is the level of certainty that the prediction is correct.

Table 5.8 Model comparison for 0.5 confidence

	YOLOv5m6 conf 0.5		YOLOv5l6 conf 0.5	
Image	**Time(ms)**	**Sign detected and confidence**	**Time(ms)**	**Sign detected**
19846	18.1	1 Regulatory max speed 40 0.54 1 Regulatory stop 0.76	23.2	1 Regulatory Stop 0.78
19870	18.4	1 Regulatory Yield 0.76 1 Parking 0.78	23.3	1 Regulatory Yield 0.74
19947	19.0	1 Regulatory 1 way 0.63 1 Regulatory yield 0.67	23.2	1 Regulatory Yield 0.77
19991	18.2	1 Regulatory max speed 40 0.58 1 No heavy goods vehicle 0.53	23.1	No Detection
20035	17.5	1 Regulatory heigh limit 1 No heavy goods vehicle	22.3	1 Regulatory height limit

The figures below show the detections as were listed in Table 5.8.

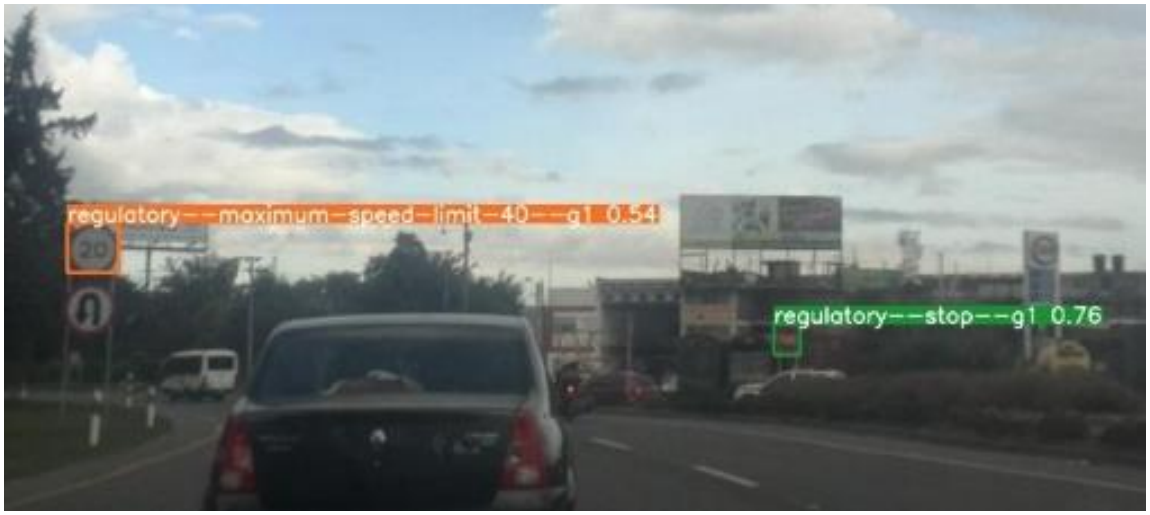


Figure 5.1 5m6 Model detection on image 19846 for 0.5 confidence

Figure 5.2 5m6 Model detection on image 19870 for 0.5 confidence

Figure 5.3 5m6 Model detection on image 19947 for 0.5 confidence

Figure 5.4 5m6 Model detection on image 19991 for 0.5 confidence

Figure 5.5 5m6 Model detection on image 20035 for 0.5 confidence

The figures below show the detections of the 5l6 model for a confidence level of 0.5. The images that don't have any detections are left out. The detections can be seen in Table 5.8. The unclear detections are highlighted on the picture with a red circle.

Figure 5.6 5l6 Model detection on image 19846 for 0.5 confidence

Figure 5.7 5l6 Model detection on image 19870 for 0.5 confidence

Figure 5.8 5l6 Model detection on image 19947 for 0.5 confidence

Figure 5.9 5l6 Model detection on image 20035 for 0.5 confidence

With the same hardware the detections are done with a confidence level of 0.75 for the same images. The results are documented in Table 5.9. The same 5m6 and 5l6 model are used.

Table 5.9 Model comparison for 0.75 confidence

	YOLOv5m6 conf 0.75		YOLOv5l6 conf 0.75	
Image	**Time(ms)**	**Sign detected**	**Time(ms)**	**Sign detected**
19846	18.7	1 Regulatory Stop 0.76	22.7	1 Regulatory Stop 0.78
19870	18.5	1 Regulatory Yield 0.76	22.9	No detection
19947	18.3	No Detection	22.7	1 Regulatory Yield 0.77
19991	18.4	No Detection	22.8	No Detection
20035	17.7	No Detection	22.9	No Detection

The figures below show the detection results of the 5m6 model with a confidence of 0.75. The images without any detections are left out.

Figure 5.10 5m6 Model detection on image 19846 for 0.75 confidence

Figure 5.11 5m6 Model detection on image 19870 for 0.75 confidence

The resulting images are shown below for the 5l6 model detection for the confidence level 0.75. The results are documented n Table 5.9.

Figure 5.12 5l6 Model detection on image 19846 for 0.75 confidence

Figure 5.13 5l6 Model detection on image 19947 for 0.75 confidence

It is difficult to compare the two models as both operate differently. Both are trained with the exact same dataset, but they evolved differently. This causes different decisions on the exact same input. The 5l6 model has more hidden layers and performs slower compared to the 5m6 model. The 5m6 model displays mode false positives than the 5l6 model and is less certain about detections. The best of the models can be achieved by tweaking the confidence level as there

are correctly identified signs at lower confidence levels. This should be done to exclude as many as possible false positives.

The largest variable is the sign size in the image. Both model's input layer is for 1280 pixels. If the image is larger than the input layer it is scaled down. This causes signs that is far in the background to shrink and become only a few pixels. This causes information to get lost. This can be seen in Figure 5.2 and Figure 5.11 where a sign is similar to another in shape and color but the information on the signs is mis represented. The problem can be solved by training a larger model. This will cause the model to perform slower and it will require higher end hardware to operate efficiently. The other solution is to grow the database and to get more images to train on. This will increase the confidence of the predictions and result in more true positives.

5.4 Web development

To display the functionality and the working of the solution, an interactive website was created to let anyone use the RSI system. The website is created so that anyone can upload an image or video and receive a file with the detections. Figure 5.14. shows the website.

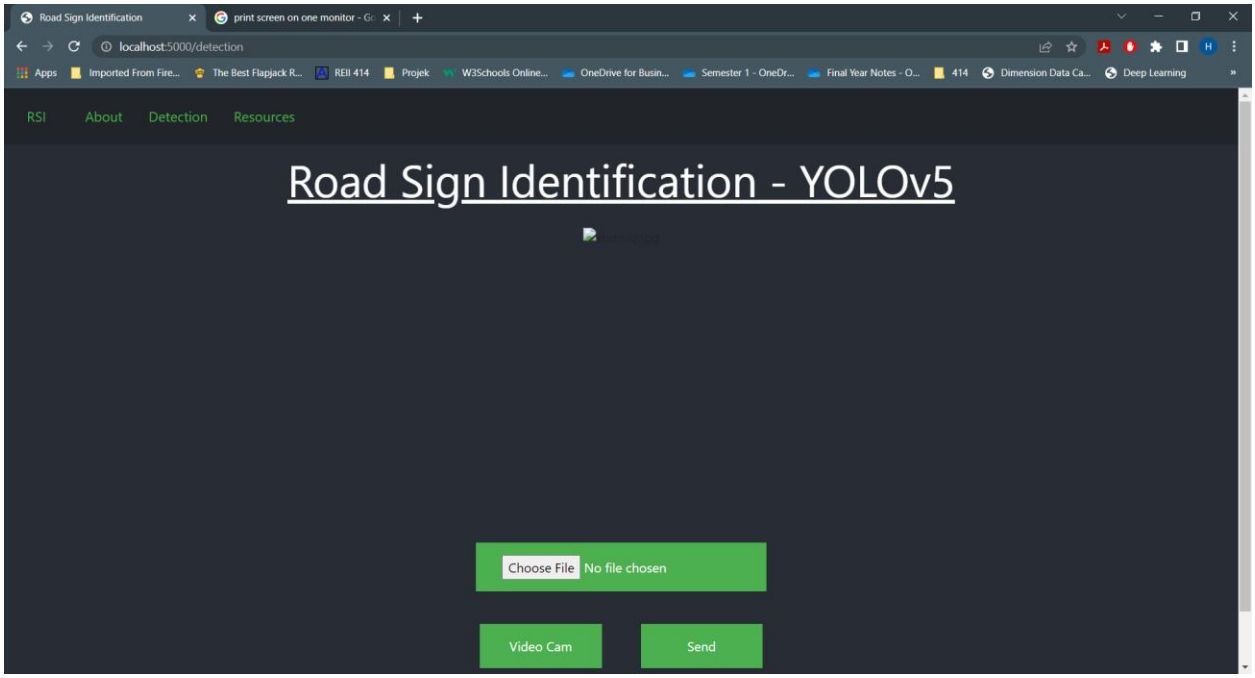


Figure 5.14 Detection page of website

The page shows a navigation bar at the top. The first link is the home page and tells the user about the project. The second page gives more detail about the project and how it works. The detection page is what is displayed in Figure 5.14. It gives the user a place to upload an image or video. The slider is used to select the desired confidence level the detections must be doen on. The button sends the image to be processed.

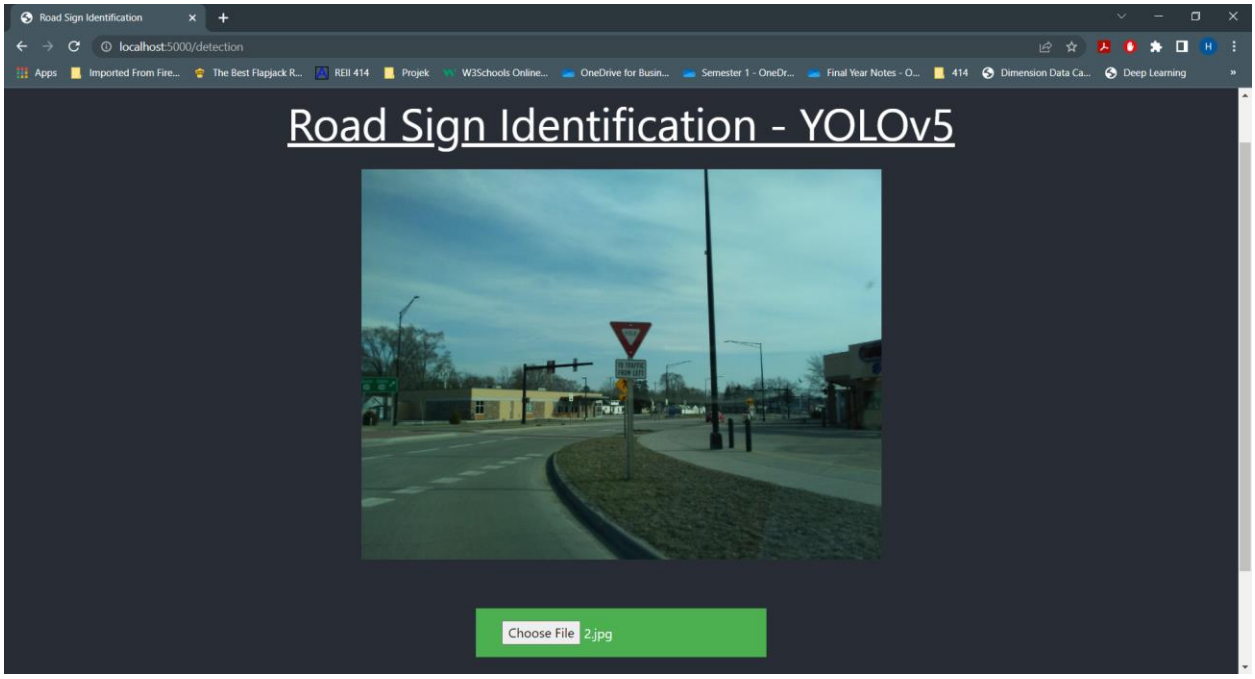


Figure 5.15 Detection page with an image loaded to be processed

Figure 5.15 shows when an image is uploaded it is displayed to be previewed by the user.

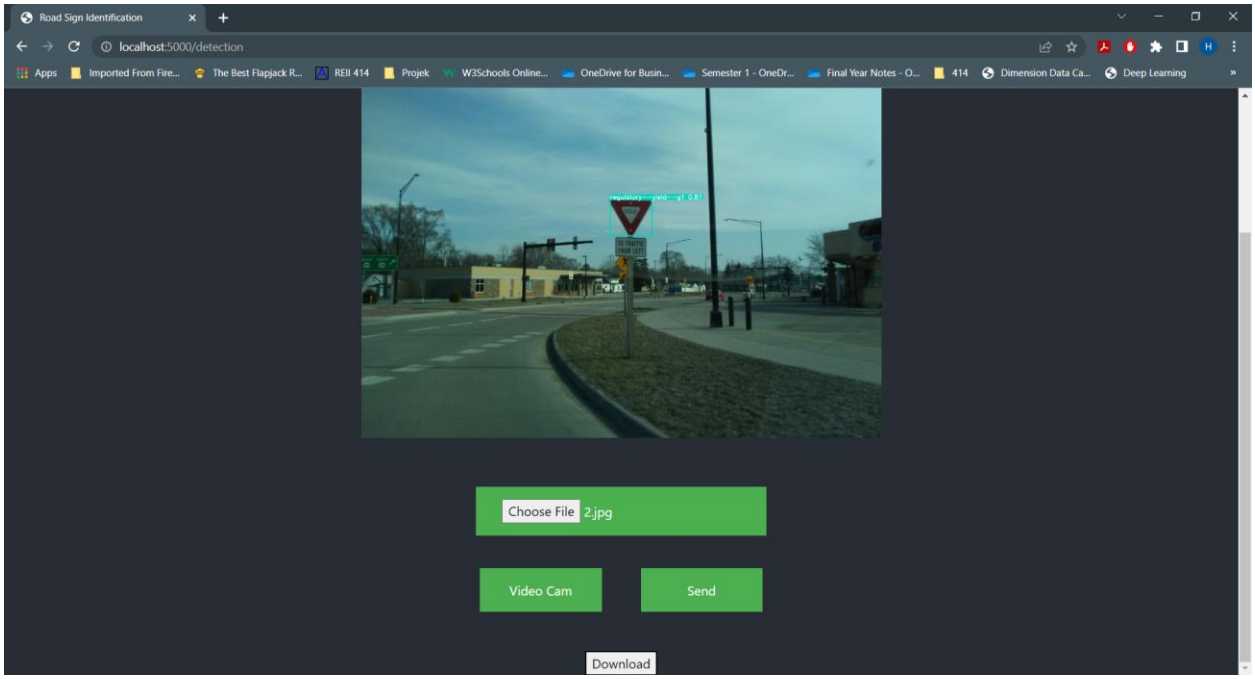


Figure 5.16 Returned image displayed with download link

Figure 5.16 shows the finished process with the resulting picture being showed to the user. There is also a link to download the image. The site works with videos as well. The code of the site is available in Appendix G.

5.5 Conclusion

The different models are trained, and the results obtained from running unseen data is compared. The 5m model which used an input layer of 320 pixels worked sub optimally. Information from the image got lost as the image resolution were scaled down dramatically. This caused signs that were in the background to shrink to smaller than 3 pixels. Detection at this size is almost impossible and it is not viable to use this model. The 5m6 model and the 5l6 model both performed well. Both were trained with a mAP of 78.45%. Both performed reasonably the same and have almost similar mistakes on detections. The 5l6 model has a higher confidence with predictions whereas the 5m6 has more correct predictions with a lower confidence. The confidence level can be used to filter out incorrect predictions. Although both models performed relatively the same the 5m6 model performed on average almost 20% faster than the 5l6 model. If a real time model is needed, then the 5m6 model is the best whereas the 5l6 model would be more suited for an application where certainty is the main priority.

CHAPTER 6 CONCLUSION AND FURTHER WORK

6.1 Conclusion

The goal of this project is to show that it is possible to develop and implement a Road Sign Identification system in South Africa. The entire world is moving to automated systems that utilizes computer vision and the automotive industry is one which can benefit from it. Existing systems are far safer than any human operator and can even predict accidents before they happen by analysing driving behaviour and comparing it to previous seen behaviour. But before the system can control a 1.5-ton car and keep the passengers within safe it first needs to start where every learner driver starts and that is with the identification of road signs.

This project is the first step into the right direction as it shows how far the technology and techniques have come. Existing networks are taken and, using a limited database, those networks are trained to detect 12 different road signs. These signs form just a fraction of the total number of road signs on South African roads, but they are the most common signs. This serves as a starting point as the networks can be evaluated as well as the possibilities with the hardware available to see what works and what needs improvements.

The student looked at the viable solutions and made the decision to use the YOLO network as it performed the best overall in previous tests. The YOLO network has many different models where the hidden layers differ. The larger models have more hidden layers whereas the smaller models have fewer hidden layers. This influences the computation time as more computations needs to be done for a larger model. The image size influences the networks input size and affects the quality of the information extracted from the image. Smaller image sizes results in information being lost, while larger image sizes affect the network size which can slow down the computation time. These attributes were changed to get the best performing network for the system.

The models which utilize smaller resolution images does not work as signs in the background gets mistaken for others due to pixels being lost during the conversion process. Signs that are up front works best with small resolution images as less image information is lost but, with a moving image such as a video or live stream limits the detection distance which could affect reaction time if the system is installed in a vehicle. The larger models which utilize larger images also struggle with images in the background, but common shapes can be identified and through generalization more accurate predictions can be made. An example of this as a yield roundabout sign being mistaken for a yield sign, although different the main objective of the sign can still be achieved as the system detects it as a yield and as the sign gets closer the correct detection can be made. Higher resolution images can be used but the computation time will increase, or more expensive hardware is needed.

Overall, the 5m6 model and the 5l6 model performed relatively the same. Both models were trained to roughly the same performance metrics and performed roughly the same with the test data. The two networks cannot be compared directly as they both differ internally. Both were trained with the same data, but the adjustments made to the nodes during the training process differ. This causes the networks to make predictions with different confidence levels. Both networks can predict the correct sign but because of the difference in confidence levels and the confidence threshold the correct sign is not shown as a detection. This can be improved by

having a larger training set with more variation in the lighting as well as the angles of the images with regards to the road sign.

Both networks performed exceptionally well considering the small dataset being used as well as the different sized images being compressed to fit the network's input layer. Concerning which network performed the best would be influenced by the application. The 5m6 model made more correct predictions at a lower confidence threshold with a faster computing speed but the network also made a few false positive detections. It is thus a lower accuracy model with a fast computation rate. The 5l6 model made less detection the true detections were at a higher confidence level but at the cost of a slower computation time. The model performs more accurately at a slower computation time. The 5m6 model can be used in a lower risk setting where computation time is important whereas the 5l6 model can be used in a live video environment where a small delay can be expected in the output.

The project satisfies the requirements set in Chapter 1.7. The network detects and recognizes the different signs it was trained on with new data. The system is also on a website to make interaction and useability easier.

6.2 Further work

There are some improvements that can be made to the project. Firstly, the database needs to be updated. This includes more images that are taken on South African roads are needed to improve the generalization of the models. This will help the model to more accurately make predictions when dealing with environmental factors such as bright days or sunsets found in South Africa. More sign classes can also be added to the database to improve the generalization of the different sign shapes and colours.

The hyperparameters can be changed while training. This was not touched when completing this project. The hyperparameters change the way the model is trained and evolves during training. This can increase its generalization as well as its training time and the accuracy of the model. This makes the network more robust against more difficult predictions and it performs better when exposed to new environments.

A custom network can be made using the knowledge gained from this project. This allows the network to be created for a standard input which reduces the loss of information in the image. This can also reduce the computation time as the network can be made smaller or more efficient.

Lastly the network can be improved by adding the descriptions and functions of the different signs to control outputs to the sign input. This system can then be implemented on a vehicular system to control the outputs of the vehicle such as the speed and steering and evaluate the performance of the system. This should be done in scale form or in a simulator as the system still needs improvement to be used in an uncontrolled environment.

APPENDIX A Dataset Files

A.1.1 Annotation from json file


```
{
  "width": 3264,
  "objects": [
    {
      "properties": {
        "ambiguous": false,
        "dummy": false,
        "direction-or-information": true,
        "barrier": false,
        "occluded": true,
        "out-of-frame": false,
        "exterior": false,
        "included": false,
        "highway": false
      },
      "bbox": {
        "xmin": 1506.890625,
        "ymin": 1726.03125,
        "ymax": 1871.859375,
        "xmax": 1629.609375
      },
      "key": "g2lmr5cyqm9yues6m599oxhygu",
      "label": "other-sign"
    },
    {
      "properties": {
        "ambiguous": false,
        "dummy": false,
        "direction-or-information": false,
        "barrier": false,
        "occluded": false,
        "out-of-frame": false,
        "exterior": false,
        "included": false,
        "highway": false
      },
      "bbox": {
        "xmin": 1799.34375,
        "ymin": 1730.21484375,
        "ymax": 1798.9453125,
        "xmax": 1867.875
      },
      "key": "r2jh7kcfm3wnl00hxf1wkw7is2",
      "label": "regulatory--stop--g1"
    },
    {
```

```
      "properties": {
        "ambiguous": false,
        "dummy": false,
        "direction-or-information": true,
        "barrier": false,
        "occluded": true,
        "out-of-frame": false,
        "exterior": false,
        "included": false,
        "highway": false
      },
      "bbox": {
        "xmin": 2837.671875,
        "ymin": 1717.06640625,
        "ymax": 1855.125,
        "xmax": 3079.921875
      },
      "key": "w70hjqcup2edzdo9qct5a69g5u",
      "label": "other-sign"
    },
    {
      "properties": {
        "ambiguous": true,
        "dummy": false,
        "direction-or-information": false,
        "barrier": false,
        "occluded": true,
        "out-of-frame": false,
        "exterior": false,
        "included": false,
        "highway": false
      },
      "bbox": {
        "xmin": 2857.59375,
        "ymin": 1853.9296875,
        "ymax": 1887.99609375,
        "xmax": 3091.078125
      },
      "key": "e7k4a35dxcmni2o92dscnb",
      "label": "other-sign"
    }
  ],
  "ispano": false,
  "height": 2448
}
```

APPENDIX B Algorithms to handle datasets

B.1.1 Class detection Algorithm

```
import json
import pandas as pd
import os
import numpy as np

#pd.set_option('display.max_colwidth', None, 'display.max_colwidth', None)

path = 'D:\\Projek Dataset\\Annotation\\mtsd_v2_partially_annotated\\annotations'

os.chdir(path)
sign = []
trouble = []

def read_json_file(file_path):
    with open(file_path, 'r') as datafile:
        data = json.load(datafile)
        df = pd.DataFrame(data)
        pf = pd.DataFrame(df)
        test = pd.DataFrame(pf['objects'].tolist(), index=pf.index)
        #print(len(test.columns))
        #print(test)
        if len(test.columns) == 0:
            trouble.append(file)
            return
        coordinates = pd.DataFrame(test['bbox'].tolist(), index=test.index)
        info = pd.DataFrame(
        {'width': df['width'], 'height': df['height'], 'label': test['label'],
'xmin': coordinates['xmin'],'ymin': coordinates['ymin'], 'xmax':
coordinates['xmax'], 'ymax': coordinates['ymax']})
        # info = info.drop(info[info.label == 'other-sign'].index)
        #print(info['label'])
        s_array = info['label'].to_numpy()
        #print(s_array)
        for x in s_array:
            if x not in sign:
                sign.append(x)


for file in os.listdir():
    if file.endswith(".json"):
        file_path = f"{path}\\{file}"
        print('File Name##########################################')
        print(file)
        read_json_file(file_path)

f = open('myfile2.txt', 'w')
```

```
for x in sign:
    f.write(x)
    f.write('\n')

f.close()

f = open('trouble2.txt', 'w')
for x in trouble:
    f.write(x)
    f.write('\n')
f.close()
```

B.1.2 Check Exist algorithm

```
import json
import os

#place where images is located
train_images = 'D:\\Projek Dataset\\Training\\images'
test_images = 'D:\\Projek Dataset\\Test\\images'
val_images = 'D:\\Projek Dataset\\Val\\images'

#place wher new files is written
new_test = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\RSI\\test'
new_train = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI\\train'
new_val = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI\\valid'
new_image_path =''


file_name = 'g'

#annotations
path = 'D:\\Projek Dataset\\Annotation\\mtsd_v2_fully_annotated\\annotations'
new_path = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI'

os.chdir(path)
trouble = []

def read_json_file(file_path):
    with open(file_path, 'r') as datafile:
        global train_images, test_images, val_images, new_image_path, new_train,
new_test, new_val, image_path

#Checks the folder of the image and sets the correct variables

        jpeg_path = os.path.splitext(file)[0]
        if os.path.exists(f"{train_images}\\{jpeg_path}"'.jpg'):
            #print(jpeg_path)
            return
```

```
        elif os.path.exists(f"{test_images}\\{jpeg_path}"'.jpg'):
            #print(jpeg_path)
            return
        elif os.path.exists(f"{val_images}\\{jpeg_path}"'.jpg'):
            #print(jpeg_path)
            return
        else:
            print('path to image does not exist elif:')
            print(file.rstrip('.json'))
            print(jpeg_path)


for file in os.listdir():
    if file.endswith(".json"):
        file_path = f"{path}\\{file}"
        read_json_file(file_path)
```

B.1.3 List of South African Road Signs extracted from the database

complementary--chevron-left--g5

complementary--one-direction-left--g1

information--motorway--g1

information--parking--g1

regulatory--height-limit--g1

regulatory--go-straight-or-turn-left--g1

regulatory--bicycles-only--g1

regulatory--maximum-speed-limit-100--g1

regulatory--maximum-speed-limit-40--g1

regulatory--maximum-speed-limit-60--g1

regulatory--no-entry--g1

regulatory--no-heavy-goods-vehicles--g1

regulatory--no-overtaking--g5

regulatory--one-way-straight--g1

regulatory--one-way-left--g1

regulatory--no-u-turn--g3

regulatory--stop--g1

regulatory--turn-left--g1

regulatory--yield--g1

warning--crossroads--g1

warning--curve-left--g1

warning--curve-right--g1

warning--double-curve-first-right--g1

warning--pedestrians-crossing--g5

warning--railroad-crossing-without-barriers--g3

warning--road-bump--g1

warning--roundabout--g1

warning--slippery-road-surface--g1

warning--uneven-road--g6

other-sign

APPENDIX C Annotation Algorithm and Files

C.1.1 Yaml file of the dataset

```
path: ../datasets/RSI  # dataset root dir

train: images/train
val: images/val
test: images/test


nc: 12
names: [ 'information--parking--g1', 'regulatory--height-limit--g1', 'regulatory--maximum-speed-limit-40--g1', 'regulatory--maximum-speed-limit-60--g1', 'regulatory--no-entry--g1', 'regulatory--no-heavy-goods-vehicles--g1', 'regulatory--one-way-straight--g1', 'regulatory--one-way-left--g1', 'regulatory--stop--g1', 'regulatory--yield--g1', 'warning--road-bump--g1', 'warning--slippery-road-surface--g1']
```

C.1.2 Annotation Conversion Algorithm

```
import json
import pandas as pd
import os
import numpy as np

#pd.set_option('display.max_colwidth', None, 'display.max_colwidth', None)

use_file = open('Use.txt', 'r') ##############################################3
use = []
for x in use_file:
    y = x.strip('\n')
    use.append(y)
print(use)
use_file.close()

trouble = []
trouble_file = open('trouble.txt', 'r')
for x in trouble_file:
    y = x.strip('\n')
    trouble.append(y)
print(trouble)
trouble_file.close()

#amount of signs in images
total_signs = np.zeros(33)
train_signs = np.zeros(33)
test_signs = np.zeros(33)
val_signs = np.zeros(33)
```

```
#place where images is located
train_images = 'D:\\Projek Dataset\\Training\\images'
test_images = 'D:\\Projek Dataset\\Test\\images'
val_images = 'D:\\Projek Dataset\\Val\\images'

#train_images = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\New
folder (2)'
#test_images = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\New
folder (4)'
#val_images = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\New folder
(5)'


#place where new files is written
new_test = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\RSI\\test'
new_train = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI\\train'
new_val = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI\\valid'
new_image_path =''

#iamge name counters
count_train = 1
count_test = 1
count_val = 1
counter = 1
file_name = 'g'

#annotations
path = 'D:\\Projek Dataset\\Annotation\\mtsd_v2_fully_annotated\\annotations'
new_path = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI'
gtr_path = 'D:\\Projek Dataset\\archive'

#path = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\New folder'
#new_path = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\RSI'
#gtr_path = 'D:\\NWU\\5de Jaar\\Jaar Vakke\\EERI 474 Projek\\Python\\New folder
(7)'

gtr_test_images = 'D:\Projek Dataset\archive\Test'

os.chdir(path)
sign = []

def read_json_file(file_path):
    with open(file_path, 'r') as datafile:
        data = json.load(datafile)
        df = pd.DataFrame(data)
        test = pd.DataFrame(df['objects'].tolist(), index=df.index)
        if len(test.columns) == 0:
            return
        coordinates = pd.DataFrame(test['bbox'].tolist(), index=test.index)
        info = pd.DataFrame(
```

```
        {'label': test['label'], 'width': df['width'], 'height': df['height'],
'xmin': coordinates['xmin'],'ymin': coordinates['ymin'], 'xmax':
coordinates['xmax'], 'ymax': coordinates['ymax']})
        info_arr = info.values.tolist()

        # check if only other signs
        count = 0
        for x in range(len(info_arr)):
            if info_arr[x][0] not in use:
                info_arr[x][0] = use[-1]

#calculate amount of other signs
            if info_arr[x][0] == use[-1]:
                count = count+1

        global train_images, test_images, val_images, new_image_path, new_train,
new_test, new_val
        global count_train, count_test, count_val
        global counter, bool

#Checks the folder of the image and sets the correct variables
        jpeg_path = os.path.splitext(file)[0]
        if os.path.exists(f"{train_images}\\{jpeg_path}"'.jpg'):
            os.chdir(train_images)
            image_path = train_images
            new_image_path = new_train
            counter = count_train
            bool = 1

        elif os.path.exists(f"{val_images}\\{jpeg_path}"'.jpg'):
            os.chdir(val_images)
            image_path = val_images
            new_image_path = new_val
            counter = count_val
            bool=3
        else:
            print('File Name##########################################')
            print(file)


#delete image if only other signs
        if count == len(info_arr):
            os.remove(f"{image_path}\\{jpeg_path}"'.jpg')
##############################################################################
            return
        else:
            for x in info_arr[:]:
                if x[0] == 'other-sign':
                    info_arr.remove(x)
```

```
#converts data to new format
        convert_to_YOLO(info_arr)

#writes the image to the new folder
        os.rename(f"{image_path}\\{jpeg_path}"'.jpg',
f"{new_image_path}\\{'images'}\\{counter}"'.jpg')#################################
########

#increase the count of the correct image
        if bool == 1:
            count_train += 1
        elif bool == 3:
            count_val += 1


def convert_to_YOLO(info):
    for x in range(len(info)):
        label = info[x][0]
        width = info[x][1]
        height = info[x][2]
        xmin = info[x][3]
        xmax = info[x][5]
        ymin = info[x][4]
        ymax = info[x][6]

        dw = 1. / (width)
        dh = 1. / (height)
        x = (xmin + xmax) / 2.0 - 1
        y = (ymin + ymax) / 2.0 - 1
        w = xmax - xmin
        h = ymax - ymin
        x = x * dw
        w = w * dw
        y = y * dh
        h = h * dh

        Yolo_format = []
        Yolo_format.append(use.index(label))
        Yolo_format.append(x)
        Yolo_format.append(y)
        Yolo_format.append(w)
        Yolo_format.append(h)


        #increase the sign count
        global total_signs, bool, train_signs, test_signs, val_signs
        total_signs[use.index(label)] += 1

        if bool == 1:
            train_signs[use.index(label)] += 1
```

```
        elif bool == 3:
            val_signs[use.index(label)] += 1

        global new_image_path
        global counter

        os.chdir(f"{new_image_path}\\{'labels'}")  ########
        f = open(f"{counter}"'.txt', 'a')
        for x in range(len(Yolo_format)):
            f.write(str(Yolo_format[x]))
            f.write(' ')
        f.write('\n')
        f.close()


for file in os.listdir():
    if file.endswith(".json"):
        file_path = f"{path}\\{file}"
        #print('File Name#########################################')
        #print(file)
        if file in trouble:
            os.remove(file_path)
        else:
            read_json_file(file_path)
            os.remove(file_path) ##############


# Training images moval
os.chdir(test_images)
new_image_path = new_test
print(test_images)
for file in os.listdir():
    jpeg_path = os.path.splitext(file)[0]
    print(jpeg_path)

    #writes the image to the new folder
    #print(f"{test_images}\\{jpeg_path}"'.jpg')
    #print(f"{new_image_path}\\{'images'}\\{count_test}"'.jpg')
    os.rename(f"{test_images}\\{jpeg_path}"'.jpg',
f"{new_image_path}\\{'images'}\\{count_test}"'.jpg')##############################
############
    count_test += 1


################################## GTSR Database
#############################################
os.chdir(gtr_path)

def index_2d(myList, v):
    for i, x in enumerate(myList):
```

```
        if v in x:
            return (i)

def convert_to_YOLO_Gt(info):

    label = info[6]
    width = info[0]
    height = info[1]
    xmin = info[2]
    xmax = info[4]
    ymin = info[3]
    ymax = info[5]

    dw = 1. / (width)
    dh = 1. / (height)
    x = (xmin + xmax) / 2.0 - 1
    y = (ymin + ymax) / 2.0 - 1
    w = xmax - xmin
    h = ymax - ymin
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh

    global Gtr_classes

    z = index_2d(Gtr_classes, label)
    gtr_label = Gtr_classes[z][1]

    use_label = use.index(gtr_label)

    Yolo_format = []
    Yolo_format.append(use_label)
    Yolo_format.append(x)
    Yolo_format.append(y)
    Yolo_format.append(w)
    Yolo_format.append(h)


    # increase the sign count
    global total_signs, bool, train_signs, test_signs, val_signs
    total_signs[use_label] += 1
    train_signs[use_label] += 1

    global new_image_path, image_path
    global counter

    os.chdir(f"{new_image_path}\\{'labels'}")  ########
    f = open(f"{counter}"'.txt', 'a')
    for x in range(len(Yolo_format)):
```

```
        f.write(str(Yolo_format[x]))
        f.write(' ')
    f.write('\n')
    f.close()

    image_path = (f"{image_path}")
    new_image_path = (f"{new_image_path}\\{'images'}\\{counter}"'.jpg')
    os.rename(image_path,new_image_path)  ########################################
#

############################################## import use classes
################################################

use_file = open('Classes.txt', 'r')

Gtr_classes = []

for line in use_file:
    currentline = line.split(",")
    col = []
    col.append(int(currentline[0]))
    x = currentline[1].strip('\n')
    x = x.lstrip(' ')
    col.append(x)
    Gtr_classes.append(col)

use_file.close()
print(Gtr_classes)

########################### Train images ###################################
os.chdir(gtr_path)
data = pd.read_csv('Train.csv')

data = data.to_numpy()

for row in range(data.shape[0]):
    image_path = data[row,[7]]
    image_path = str(image_path).lstrip('[').rstrip(']')
    image_path = str(image_path).lstrip("'").rstrip("'")
    image_path = np.char.replace(image_path, '/', '\\')
    image_path = f"{gtr_path}\\{image_path}"


    if os.path.exists(image_path):
        new_image_path = new_train
        counter = count_train
        convert_to_YOLO_Gt(data[row][:7])
        count_train += 1
```

```
############################ Validation images
#############################################
#Move test as validation images
os.chdir(gtr_path)
data = pd.read_csv('Test.csv')
print(data)
data = data.to_numpy()
use_classes_gtr = [3, 7, 13, 14, 16, 17, 22, 23, 34, 35, 37]

for row in range(data.shape[0]):
    class_gtr = data[row, 6]
    image_path = data[row,[7]]
    image_path = str(image_path).lstrip('[').rstrip(']')
    image_path = str(image_path).lstrip("'").rstrip("'")
    image_path = np.char.replace(image_path, '/', '\\')
    image_path = f"{gtr_path}\\{image_path}"
    print(class_gtr)
    print(image_path)
    if (os.path.exists(image_path) & (class_gtr in use_classes_gtr)):
        new_image_path = new_val
        print(new_image_path)
        counter = count_val
        convert_to_YOLO_Gt(data[row][:7])
        count_val += 1
    else:
        os.remove(image_path)



############################################ write files
#############################################

os.chdir(new_path)
use.pop()
np.delete(train_signs, -1)
np.delete(test_signs, -1)
np.delete(val_signs, -1)

os.chdir(new_path)
f = open('data.yaml', 'w')
f.write('path: ' + new_path)
f.write('\ntrain: ' + new_train + '\images')
f.write('\ntest: '+ new_test + '\images')
f.write('\nval: '+ new_val + '\images')
f.write('\n\nnc: '+ str(len(use)))
f.write('\nnames: [')
for x in range(1, len(use)):
    f.write(use[x] + ', ')
f.write(str(use[-1]) + ']')
f.close()
```

```
count = []

for x in range(len(total_signs)):
    count.append(x)
col_format = '{:<20}' * 6 + '\n'
os.chdir(new_path)
f = open('all.txt', 'w')
f.write('Total Train :' + str(count_train-1)+'\n')
f.write('Total Test :' + str(count_test-1)+'\n')
f.write('Total Val :' + str(count_val-1)+'\n\n')
f.write('Number:\t\tTraining:\t\tTest\t\tValidation\t\tTotal:\t\tSign type\n\n')
for x in zip(count, train_signs, test_signs, val_signs, total_signs, use):
    f.write(col_format.format(*x))
f.close()
```

APPENDIX D YOLOv5 model architectures

D.1 YOLOv5m6 model architecture

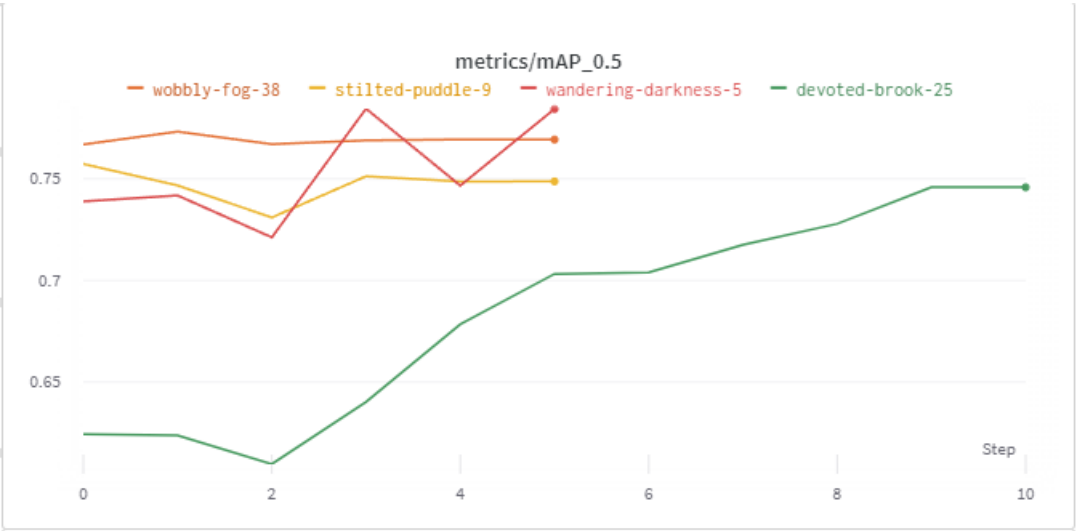
```
                 from  n    params  module                                  arguments
  0                -1  1      5280  models.common.Conv                      [3, 48, 6, 2, 2]
  1                -1  1     41664  models.common.Conv                      [48, 96, 3, 2]
  2                -1  2     65280  models.common.C3                        [96, 96, 2]
  3                -1  1    166272  models.common.Conv                      [96, 192, 3, 2]
  4                -1  4    444672  models.common.C3                        [192, 192, 4]
  5                -1  1    664320  models.common.Conv                      [192, 384, 3, 2]
  6                -1  6   2512896  models.common.C3                        [384, 384, 6]
  7                -1  1   1991808  models.common.Conv                      [384, 576, 3, 2]
  8                -1  2   2327040  models.common.C3                        [576, 576, 2]
  9                -1  1   3982848  models.common.Conv                      [576, 768, 3, 2]
 10                -1  2   4134912  models.common.C3                        [768, 768, 2]
 11                -1  1   1476864  models.common.SPPF                      [768, 768, 5]
 12                -1  1    443520  models.common.Conv                      [768, 576, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 14          [-1, 8]  1         0  models.common.Concat                    [1]
 15                -1  2   2658816  models.common.C3                        [1152, 576, 2, False]
 16                -1  1    221952  models.common.Conv                      [576, 384, 1, 1]
 17                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 18          [-1, 6]  1         0  models.common.Concat                    [1]
 19                -1  2   1182720  models.common.C3                        [768, 384, 2, False]
 20                -1  1     74112  models.common.Conv                      [384, 192, 1, 1]
 21                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 22          [-1, 4]  1         0  models.common.Concat                    [1]
 23                -1  2    296448  models.common.C3                        [384, 192, 2, False]
 24                -1  1    332160  models.common.Conv                      [192, 192, 3, 2]
 25         [-1, 20]  1         0  models.common.Concat                    [1]
 26                -1  2   1035264  models.common.C3                        [384, 384, 2, False]
 27                -1  1   1327872  models.common.Conv                      [384, 384, 3, 2]
 28         [-1, 16]  1         0  models.common.Concat                    [1]
 29                -1  2   2437632  models.common.C3                        [768, 576, 2, False]
 30                -1  1   2987136  models.common.Conv                      [576, 576, 3, 2]
 31         [-1, 12]  1         0  models.common.Concat                    [1]
 32                -1  2   4429824  models.common.C3                        [1152, 768, 2, False]
 33  [23, 26, 29, 32]  1     98124  models.yolo.Detect                      [12, [[19, 27, 44, 40, 38, 94],
[96, 68, 86, 152, 180, 137], [140, 301, 303, 264, 238, 542], [436, 615, 739, 380, 925, 792]], [192, 384, 576, 768]]
Model summary: 379 layers, 35339436 parameters, 35339436 gradients, 49.4 GFLOPs
```

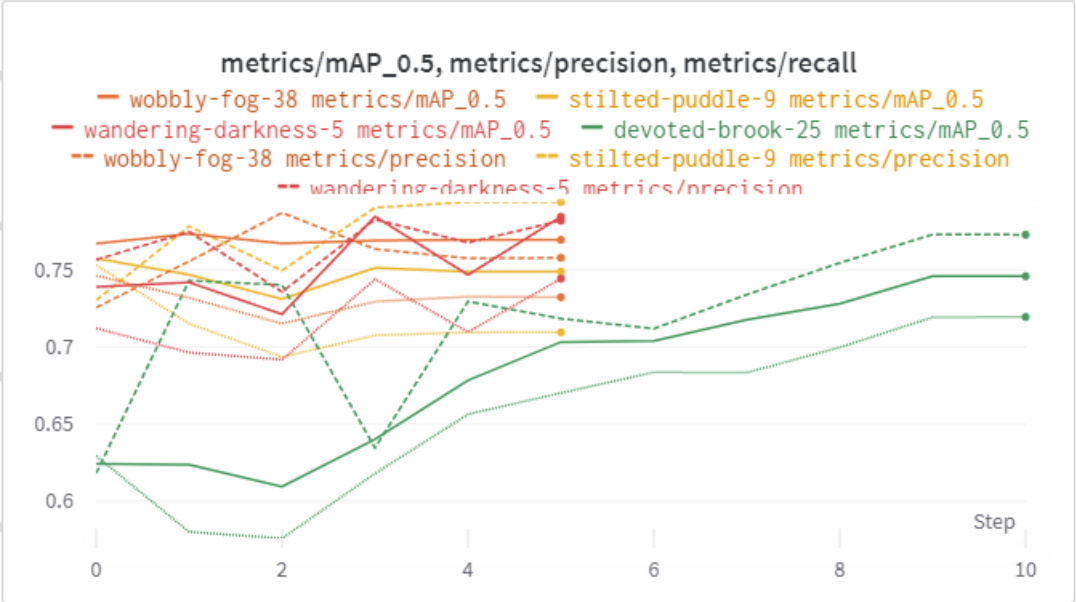
D.2 YOLOv5l6 model architecture

```
                from  n    params  module                                  arguments
  0                -1  1      7040  models.common.Conv                      [3, 64, 6, 2, 2]
  1                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  2                -1  3    156928  models.common.C3                        [128, 128, 3]
  3                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  4                -1  6   1118208  models.common.C3                        [256, 256, 6]
  5                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  6                -1  9   6433792  models.common.C3                        [512, 512, 9]
  7                -1  1   3540480  models.common.Conv                      [512, 768, 3, 2]
  8                -1  3   5611008  models.common.C3                        [768, 768, 3]
  9                -1  1   7079936  models.common.Conv                      [768, 1024, 3, 2]
 10                -1  3   9971712  models.common.C3                        [1024, 1024, 3]
 11                -1  1   2624512  models.common.SPPF                      [1024, 1024, 5]
 12                -1  1    787968  models.common.Conv                      [1024, 768, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 14          [-1, 8]  1         0  models.common.Concat                    [1]
 15                -1  3   6200832  models.common.C3                        [1536, 768, 3, False]
 16                -1  1    394240  models.common.Conv                      [768, 512, 1, 1]
 17                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 18          [-1, 6]  1         0  models.common.Concat                    [1]
 19                -1  3   2757632  models.common.C3                        [1024, 512, 3, False]
 20                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 21                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 22          [-1, 4]  1         0  models.common.Concat                    [1]
 23                -1  3    690688  models.common.C3                        [512, 256, 3, False]
 24                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 25          [-1, 20]  1         0  models.common.Concat                    [1]
 26                -1  3   2495488  models.common.C3                        [512, 512, 3, False]
 27                -1  1   2360320  models.common.Conv                      [512, 512, 3, 2]
 28          [-1, 16]  1         0  models.common.Concat                    [1]
 29                -1  3   5807616  models.common.C3                        [1024, 768, 3, False]
 30                -1  1   5309952  models.common.Conv                      [768, 768, 3, 2]
 31          [-1, 12]  1         0  models.common.Concat                    [1]
 32                -1  3  10496000  models.common.C3                        [1536, 1024, 3, False]
 33  [23, 26, 29, 32]  1    130764  models.yolo.Detect                      [12, [[19, 27, 44, 40, 38, 94],
[96, 68, 86, 152, 180, 137], [140, 301, 303, 264, 238, 542], [436, 615, 739, 380, 925, 792]], [256, 512, 768, 1024]]
Model summary: 477 layers, 76247116 parameters, 76247116 gradients, 110.7 GFLOPs
```

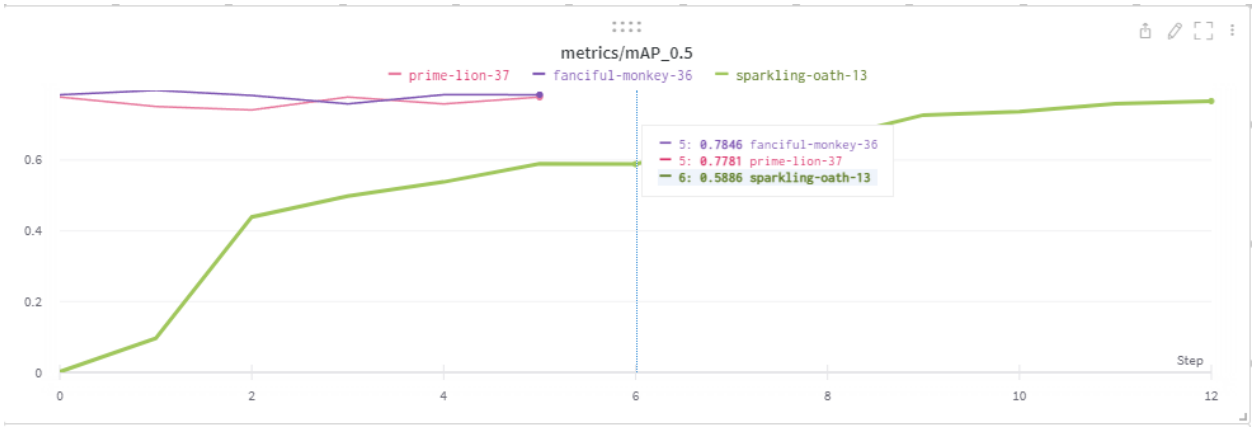
APPENDIX E Training

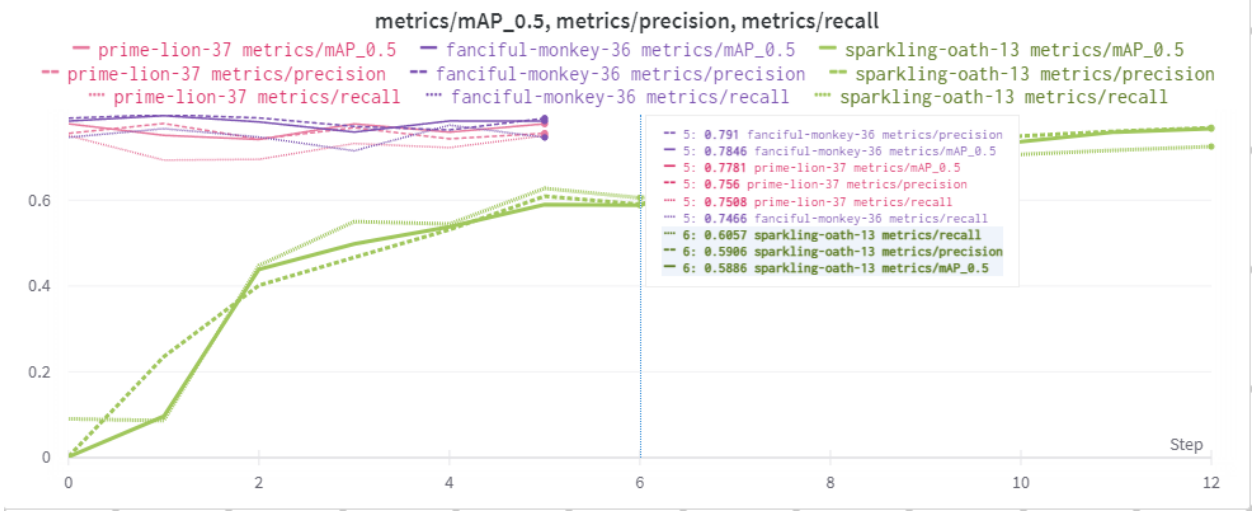
E.1 Training runs performance metrics for YOLOv5m6 model

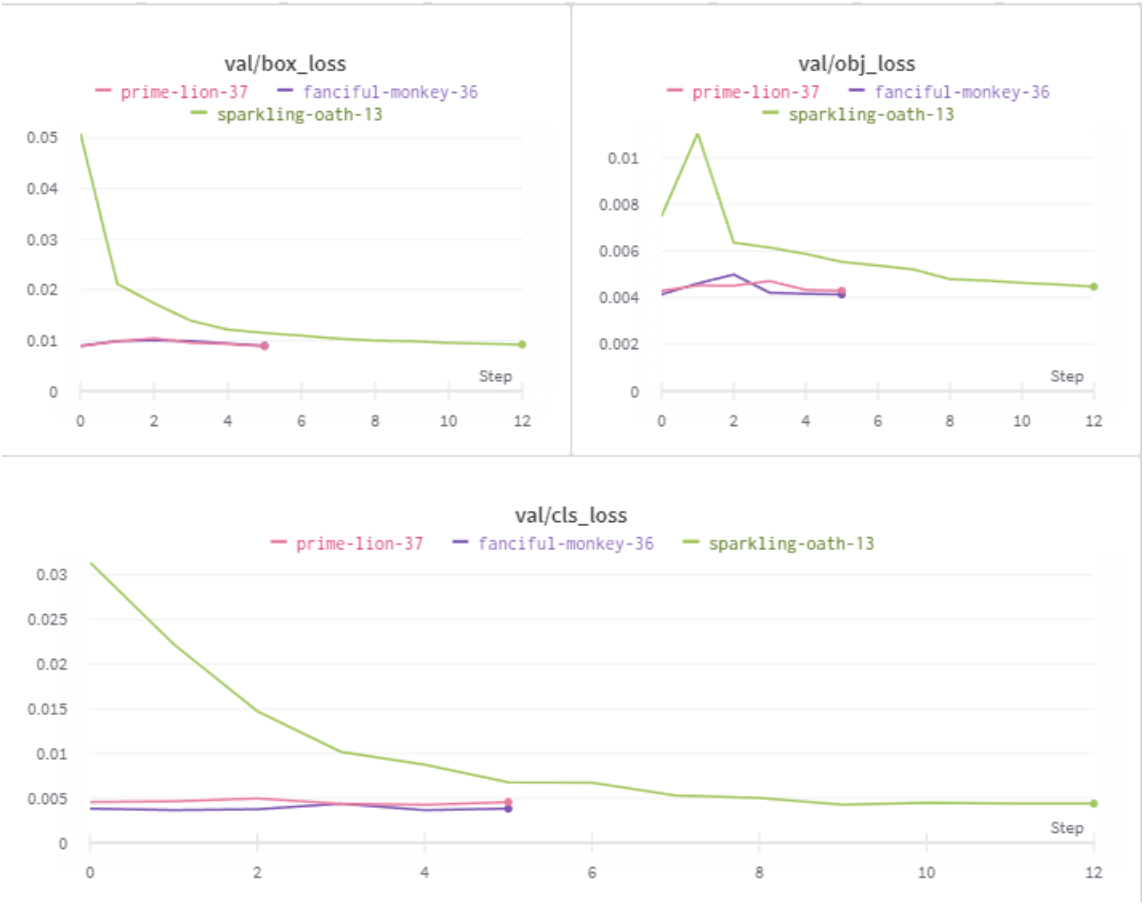




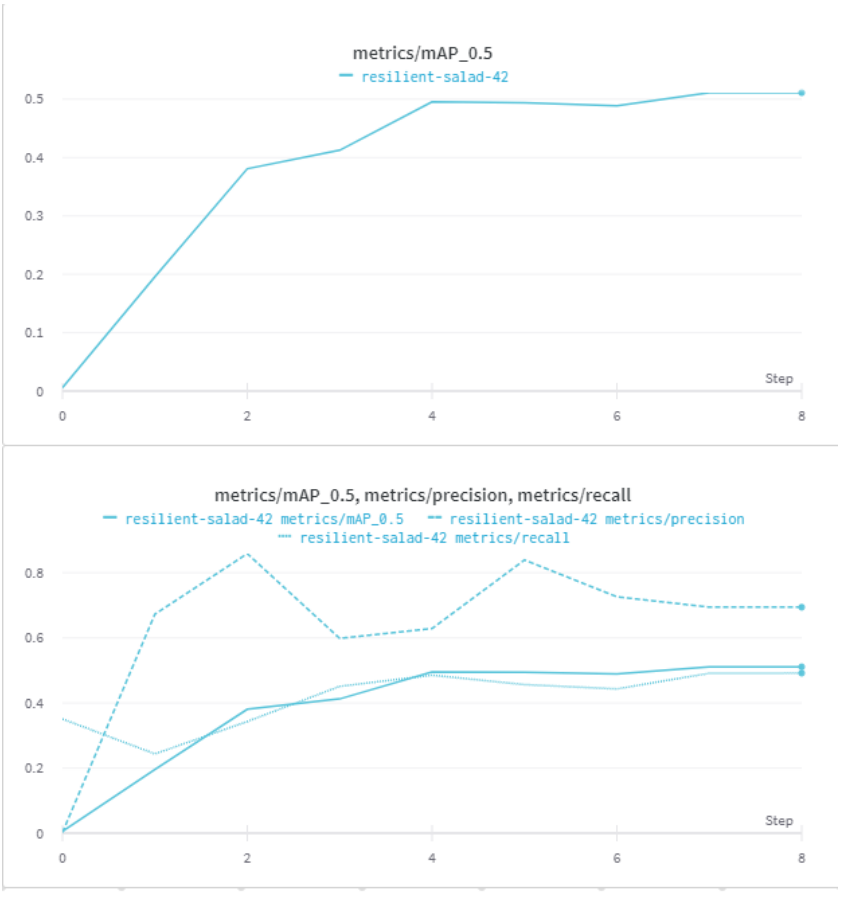
E.2 Training runs performance metrics for YOLOv5l6 model

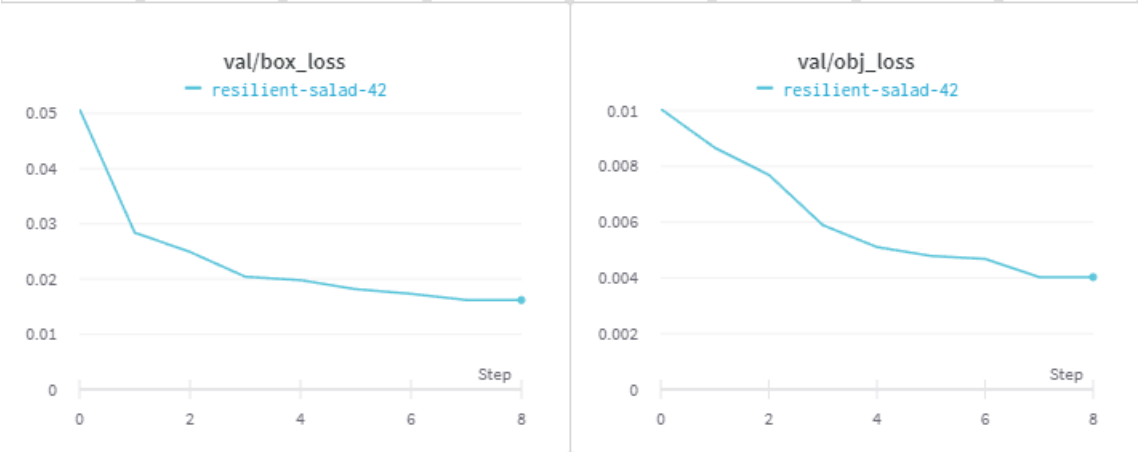


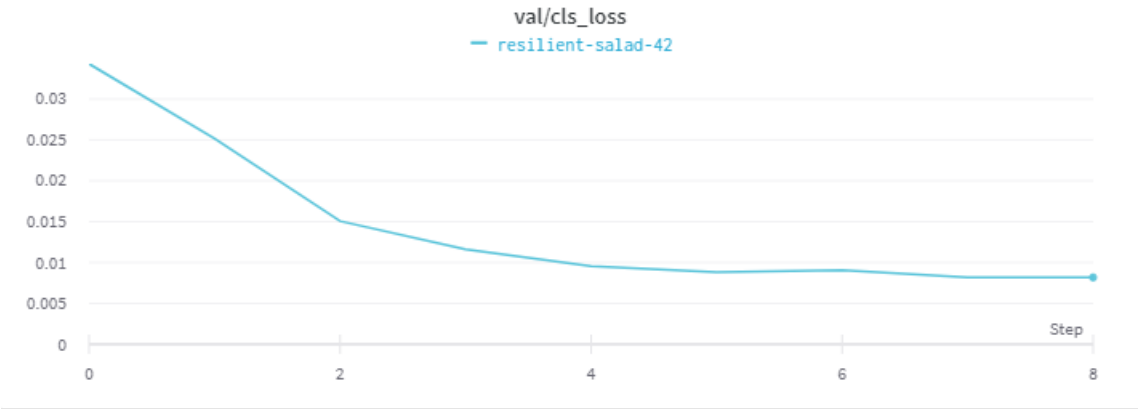




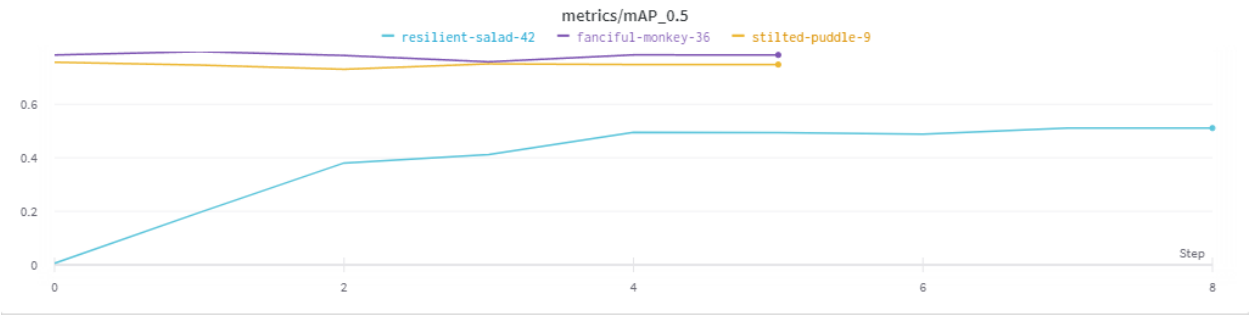
E.3 Training runs performance metrics YOLOv5l6 compared with YOLOv5m

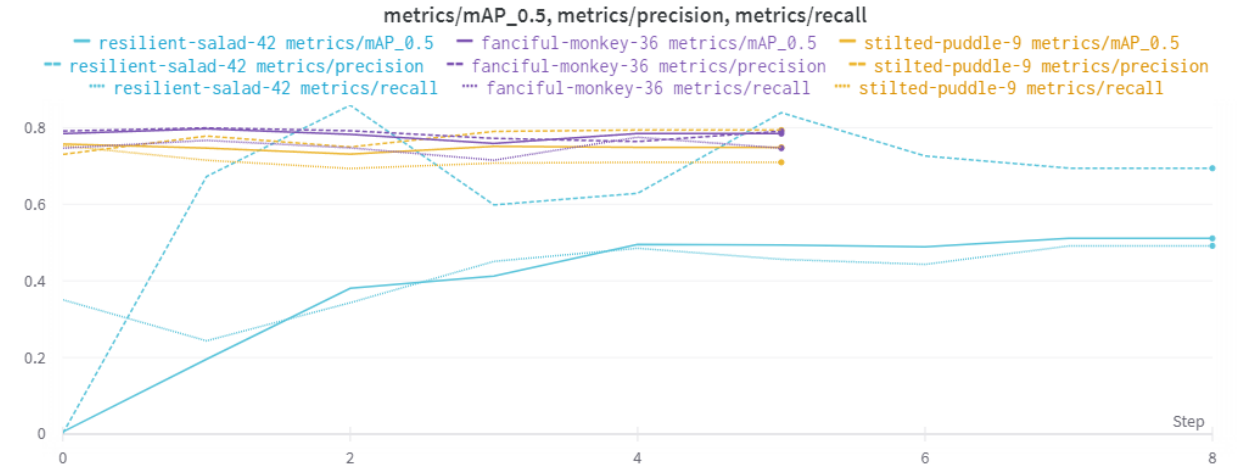


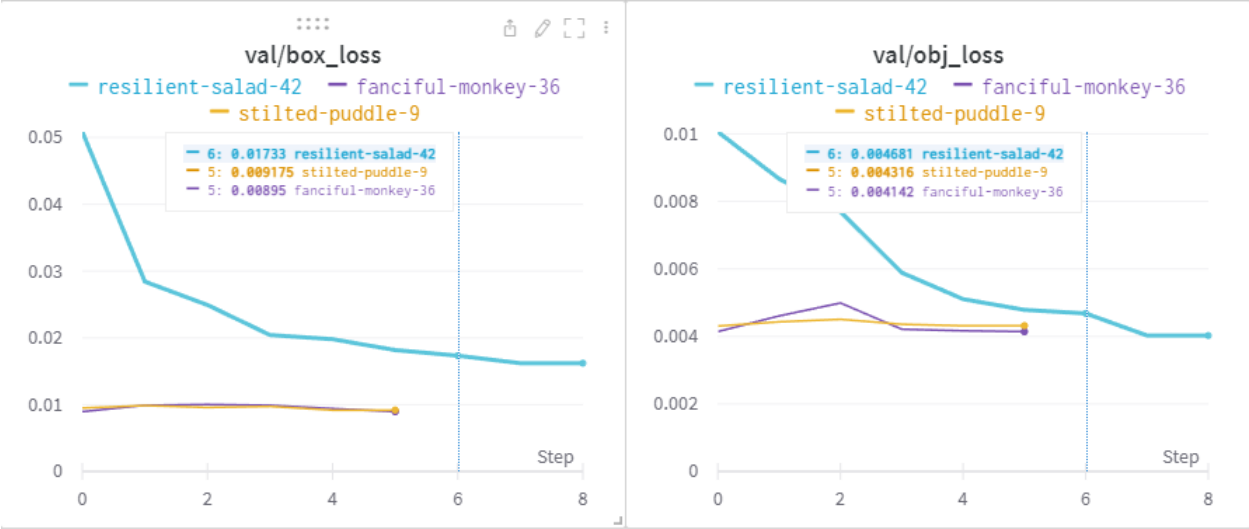




E.4 YOLO models 5m6, 5l6 and 5m best runs compared






val/box_loss
resilient-salad-42
fanciful-monkey-36
stilted-puddle-9
6: 0.01733 resilient-salad-42
5: 0.009175 stilted-puddle-9
5: 0.00895 fanciful-monkey-36
0.05
0.04
0.03
0.02
0.01
0
Step
0
2
4
6
8

val/obj_loss
resilient-salad-42
fanciful-monkey-36
stilted-puddle-9
6: 0.004681 resilient-salad-42
5: 0.004316 stilted-puddle-9
5: 0.004142 fanciful-monkey-36
0.01
0.008
0.006
0.004
0.002
0
Step
0
2
4
6
8


val/cls_loss
resilient-salad-42
fanciful-monkey-36
stilted-puddle-9
6: 0.009049 resilient-salad-42
5: 0.004219 stilted-puddle-9
5: 0.003819 fanciful-monkey-36
0.03
0.025
0.02
0.015
0.01
0.005
0
Step
0
2
4
6
8

APPENDIX F Images used to test models


Theatergarage

APPENDIX G Website code

G.1 Python document for website

```
from re import DEBUG, sub
from flask import Flask, render_template, request, redirect, send_file, url_for
from werkzeug.utils import secure_filename, send_from_directory
import os
import subprocess

app = Flask(__name__)
#nav = Navigation(app)


uploads_dir = os.path.join(app.instance_path, 'uploads')

os.makedirs(uploads_dir, exist_ok=True)


@app.route("/")
def index():
    return render_template('index.html')

@app.route("/about")
def about():
    return render_template("about.html");

@app.route("/detection")
def detection():
    return render_template("detection.html");

@app.route("/resources")
def resources():
    return render_template("resources.html");


@app.route("/detect", methods=['POST'])
def detect():
    print('hello')
    if not request.method == "POST":
        return
    conf = request.form['conf']
    print(conf)

    video = request.files['video']
    video.save(os.path.join(uploads_dir, secure_filename(video.filename)))
    print('hellok')
    print(video)
    print(conf)
```

```
    #subprocess.run("ls", shell=False)
    subprocess.run(['py', 'detect.py', '--source', os.path.join(uploads_dir,
secure_filename(video.filename)), '--data', 'RSI_data.yaml', '--weights',
'rsi_weight.pt', '--conf', conf, '--img', '1280'], shell=False)
    # return os.path.join(uploads_dir, secure_filename(video.filename))
    obj = secure_filename(video.filename)
    return obj

@app.route("/opencam", methods=['GET'])
def opencam():
    print("here")
    subprocess.run(['py', 'detect.py', '--source', '0', '--data',
'RSI_data.yaml', '--weights', 'rsi_weight.pt', '--img', '1280'], shell=False) #'-
-data', 'data/RSI_data.yaml',
    return "done"


@app.route('/return-files', methods=['GET'])
def return_file():
    obj = request.args.get('obj')
    loc = os.path.join("runs/detect", obj)
    print(loc)
    try:
        return send_file(os.path.join("runs/detect", obj),
attachment_filename=obj)
        # return send_from_directory(loc, obj)
    except Exception as e:
        return str(e)
```

G.2 Python detection document

```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license
"""
Run YOLOv5 detection inference on images, videos, directories, globs, YouTube,
webcam, streams, etc.

Usage - sources:
    $ python detect.py --weights yolov5s.pt --source
0                               # webcam
                                                    img.jpg
    # image
                                                    vid.mp4
    # video
                                                    path/
    # directory
                                                    'path/*.jpg'
   # glob
```

```
                                                    'https://youtu.be/Zgi9g1ksQH
c'  # YouTube
                                                    'rtsp://example.com/media.mp
4'  # RTSP, RTMP, HTTP stream

Usage - formats:
    $ python detect.py --weights yolov5s.pt                 # PyTorch
                                 yolov5s.torchscript        # TorchScript
                                 yolov5s.onnx               # ONNX Runtime or
OpenCV DNN with --dnn
                                 yolov5s.xml                # OpenVINO
                                 yolov5s.engine             # TensorRT
                                 yolov5s.mlmodel            # CoreML (macOS-only)
                                 yolov5s_saved_model        # TensorFlow
SavedModel
                                 yolov5s.pb                 # TensorFlow GraphDef
                                 yolov5s.tflite             # TensorFlow Lite
                                 yolov5s_edgetpu.tflite     # TensorFlow Edge TPU
                                 yolov5s_paddle_model       # PaddlePaddle
                                 exp
"""

import argparse
import os
import platform
import sys
from pathlib import Path

import torch

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages,
LoadScreenshots, LoadStreams
from utils.general import (LOGGER, Profile, check_file, check_img_size,
check_imshow, check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args,
scale_boxes, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, smart_inference_mode


@smart_inference_mode()
def run(
        weights=ROOT / 'yolov5s.pt',  # model path or triton URL
```

```
        source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=True,  # show results
        save_txt=True,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        vid_stride=1,  # video frame-rate stride
):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference
images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://',
'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not
is_file)
    screenshot = source.lower().startswith('screen')
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_txt = True
    #save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  #
increment run
    save_dir = Path('static')  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
```

```
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data,
fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img = check_imshow()
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
    for path, im, im0s, vid_cap, s in dataset:
        with dt[0]:
            im = torch.from_numpy(im).to(model.device)
            im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
visualize else False
            pred = model(im, augment=augment, visualize=visualize)

        # NMS
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
```

```
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode ==
'image' else f'_{frame}')  # im.txt
            result_txt = str(save_dir / 'labels' / 'result')
            s += '%gx%g ' % im.shape[2:]  # print string
            outputter = '' # print
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4],
im0.shape).round()


                with open(f'{result_txt}.txt', 'a') as f:
                    # Print results
                    for c in det[:, 5].unique():
                        n = (det[:, 5] == c).sum()  # detections per class
                        outputter += f"{n} {names[int(0)]}{'s' * (n > 1)}, "
                        s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to
string


                        f.write(outputter)
                    f.write(f"{dt[1].dt * 1E3:.1f}ms")

                print(outputter)
                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls,
*xywh)  # label format
                        with open(f'{txt_path}.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')
                            f.write(f"{dt[1].dt * 1E3:.1f}ms")

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf
else f'{names[c]} {conf:.2f}')
                        annotator.box_label(xyxy, label, color=colors(c, True))
                    if save_crop:
```

```
                        save_one_box(xyxy, imc, file=save_dir / 'crops' /
names[c] / f'{p.stem}.jpg', BGR=True)

            # Stream results
            im0 = annotator.result()
            if view_img:
                if platform.system() == 'Linux' and p not in windows:
                    windows.append(p)
                    cv2.namedWindow(str(p), cv2.WINDOW_NORMAL |
cv2.WINDOW_KEEPRATIO)  # allow window resize (Linux)
                    cv2.resizeWindow(str(p), im0.shape[1], im0.shape[0])
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path[i] != save_path:  # new video
                        vid_path[i] = save_path
                        if isinstance(vid_writer[i], cv2.VideoWriter):
                            vid_writer[i].release()  # release previous video
writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path = str(Path(save_path).with_suffix('.mp4'))  #
force *.mp4 suffix on results videos
                        vid_writer[i] = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer[i].write(im0)

        # Print time (inference-only)
        print(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt *
1E3:.1f}ms")
        LOGGER.info(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt *
1E3:.1f}ms")

    # Print results
    t = tuple(x.t / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per
image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
{save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
```

```
    if update:
        strip_optimizer(weights[0])  # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT /
'yolov5s.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images',
help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml',
help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int,
default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25,
help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU
threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum
detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to
*.txt')
    parser.add_argument('--save-conf', action='store_true', help='save
confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped
prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save
images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class:
--classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-
agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented
inference')
    parser.add_argument('--visualize', action='store_true', help='visualize
features')
    parser.add_argument('--update', action='store_true', help='update all
models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save
results to project/name')
    parser.add_argument('--name', default='exp', help='save results to
project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding
box thickness (pixels)')
```

```
    parser.add_argument('--hide-labels', default=False, action='store_true',
help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true',
help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-
precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-
rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt


def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

G.3 Javascript Document

```
window.onload = () => {
  var slider = document.getElementById("confidence")
  console.log(slider.value)
  $("#sendbutton").click(() => {
    console.log(slider.value)
    imagebox = $("#imagebox");
    link = $("#link");
    input = $("#imageinput")[0];
    if (input.files && input.files[0]) {
      let formData = new FormData();
      formData.append("video", input.files[0]);
      formData.append("conf", slider.value )
      console.log(formData.entries)
      $.ajax({
        url: "/detect", // fix this to your liking
        type: "POST",
        data: formData,
        cache: false,
        processData: false,
        contentType: false,
```

```
        error: function (data) {
          console.log("upload error", data);
          console.log(data.getAllResponseHeaders());
        },
        success: function (data) {
          console.log(data);
          //show_image(data, 480, 480, Sign);
          // bytestring = data["status"];
          // image = bytestring.split("'")[1];
          $("#link").css("visibility", "visible");
          $("#download").attr("href", "static/" + data);
          $("#image").attr("src", "static/" + data, "href", "static/" + data);
          show_image(data, 480, 480, Sign);
        },
      });
    }
  });
  $("#opencam").click(() => {
    console.log("evoked openCam");
    $.ajax({
      url: "/opencam",
      type: "GET",
      error: function (data) {
        console.log("upload error", data);
      },
      success: function (data) {
        console.log(data);
      }
    });
  })
};

function readUrl(input) {
  imagebox = $("#imagebox");
  console.log(imagebox);
  console.log("evoked readUrl");
  if (input.files && input.files[0]) {
    let reader = new FileReader();
    reader.onload = function (e) {
      console.log(e.target);

      imagebox.attr("src", e.target.result);
      $('#image').attr('src', e.target.result);
      $('#blah').attr('src', e.target.result);
      //   imagebox.height(500);
      //   imagebox.width(800);
    };
    reader.readAsDataURL(input.files[0]);
  }
}
```

```
function openCam(e){
  console.log("evoked openCam");
  e.preventDefault();
  console.log("evoked openCam");
  console.log(e);
}

function show_image(src, width, height, alt) {
  var img = document.createElement("img");
  img.src = src;
  img.width = width;
  img.height = height;
  img.alt = alt;

  // This next line will just add it to the <body> tag
  document.body.appendChild(img);
}
```

REFERENCES

7

[1] D. K. Foote, “A Brief History of Deep Learning,” Dataversity, 7 02 2017. [Online]. Available: https://www.dataversity.net/brief-history-deep-learning/#:~:text=The%20history%20of%20Deep%20Learning,to%20mimic%20the%20thought%20process.. [Accessed 12 02 2022].

[2] “History of Autonomous cars,” Tomorrow's world today, [Online]. Available: https://www.tomorrowsworldtoday.com/2021/08/09/history-of-autonomous-cars/#:~:text=At%20the%201939%20World's%20Fair,first%20self%2Ddriving%20car%20model.&text=This%20was%20the%20first%20use,process%20images%20of%20the%20road.. [Accessed 12 02 2022].

[3] S. Papert, “The Summer Vision Project,” Massachusetts Institute of Technology, Boston, 1966.

[4] The Franklin Institute, “The Franklin Institute Awards,” The Franklin Institute , 2021. [Online]. Available: https://www.fi.edu/laureates/kunihiko-fukushima. [Accessed 28 07 2022].

[5] A. Kayy, “Artificial Neural Networks,” ComputerWorld, 12 02 2001. [Online]. Available: https://www.computerworld.com/article/2591759/artificial-neural-networks.html. [Accessed 28 07 2022].

[6] S. K. Mitra, Digital Signal Procecessing, New Delhi: Tata McGraw-Hill Publishing Co. Ltd, 1997.

[7] “Image Processing 101 Chapter 1.2: Color Models,” Dynamsoft, 17 05 2019. [Online]. Available: https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-models/. [Accessed 28 07 2022].

[8] U. Zakir, “Automatic Road Sign Detection and,” Usman Zakir 2011, Loughborough, 2011.

[9] B. Saha, D. Davies and A. Raghavan, “Day night classification of images using thresholding on HSV histogram”. USA Patent US 9,530,056 B2, 15 06 2012.

[10] A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, Dive into Deep Learning, 2019.

[11] E. Burns, “What is artificial intelligence (AI)?,” SearchEnterpriseAI, [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence#:~:text=Artificial%20intelligence%20is%20the%20simulation,speech%20recognition%20and%20machine%20vision.. [Accessed 02 08 2022].

[12] P. Grieve, “Deep learning vs. machine learning: What’s the difference?,” Zendesk Blog, 08 03 2022. [Online]. Available: https://www.zendesk.com/blog/machine-learning-and-deep-learning/. [Accessed 02 08 2022].

[13] F. Chollet, Deep learning with Python, Shelter island: Manning, 2018.

[14] R. Girshick, J. Donahue, T. Darrell, J. Maik and U. Berkley, “Rich feature hierarchies for accurate object detection and semantic segmentation.,” Proceedings of the IEEE conference on computer vision and pattern recognition, 2014.

[15] R. Girshick, “Fast R-CNN,” IEEE, 2015.

[16] J. Redmond, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once:,” University of Washington, Washington, 2016.

[17] J. Redmond and A. Farhadi, “YOLOv3: An Incremental Improvement,” University of Washington, Washington, 2018.

[18] Bandyopadhyay, Hmirshav;, “YOLO: Real-Time Object Detection Explained,” v7labs, 19 07 2022. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection. [Accessed 26 08 2022].

[19] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” in *European conference on computer vision (pp. 21–37).*, 2016.

[21] A. E. Maxwell, T. A. Warner and L. A. Guillèn, “Accuracy Assessment in Convolutional Neural Network-Based Deep Learning Remote Sensing Studies—Part 1: Literature Review,” *Remote Sensing,* vol. 13, no. 13, p. 2450, June 2021.

[22] Y.-L. Kuo and S.-H. Lin, “Applications of Deep Learning to Road Sign,” Xplore, Taipei, Taiwan.

[23] V. N. Sichkar, “ Real time detection and classification of traffic signs based on YOLO version 3 algorithm,” *Scientific and Technical Journal of Information Technologies, Mechanics and Optics,* vol. 20, no. 3, pp. 418-424, 2020.

[24] M. Li, Z. Zhang, L. Lei, X. Wang and X. Guo, “Agricultural Greenhouses Detection in High-Resolution Satellite Images Based on Convolutional Neural Networks: Comparison of Faster R-CNN, YOLO v3 and SSD,” *Sensors,* vol. 20, no. 17, p. 4938, 2020.

[25] L. Chen, Z. Liming and J. Liu, “Aircraft Recognition from Remote Sensing Images Based on Machine Vision,” *Journal of Information Processing Systems,* vol. 16, no. 4, pp. 795-808, 2020.

[26] J.-a. Kim, J.-Y. Sung and S.-H. Park, “Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition,” *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia),* pp. 1-4, 2020.

[27] Z. Ge, S. Liu, F. Wang, Z. Li and J. Sun, “YOLOX: Exceeding YOLO Series in 2021,” arXiv, 2021.

[28] U. Nepal and H. Eslamiat, “Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous,” *sensors,* vol. 22, p. 464, 2022.

[29] R. A, V. G. Pola, D. A. B. Vaishnavi and S. S. Karra, “Comparison of YOLOv3, YOLOv4 and YOLOv5 Performance for Detection of Blood Cells,” *International Research Journal of Engineering and Technology (IRJET) ,* vol. 8, no. 4, pp. 4225-4229, 2021.

[30] S. M. Ali, “Comparative Analysis of YOLOv3, YOLOv4 and YOLOv5 for Sign Language Detection,” *International Journal of Advanced Research and Innovative Ideas In Education (IJARIIE),* vol. 7, no. 4, pp. 2393-2398, 2021.

[31] P. D. D. Pitawela and L. Ranathunga, “Low Latency Approach in Road Sign Recognition,” IEEE, Katubedda, Sri Lanka, 2018.

[32] M. Ahsan, S. Das, S. Kumar and Z. L. Tasriba, “A Detailed Study on Bangladeshi Road Sign,” IEEE, Khulna, Bangladesh, 2019.

[33] J. D. Zhao, Z. M. Bai and H. B. Chen, “Research on Road Traffic Sign Recognition Based on Video Image,” CPS, Beijing, China, 2017.

[34] A. Z. Syaharuddin, Z. Zainuddin and Andani, “Multi-Pole Road Sign Detection Based on Faster Region-based Convolutional Neural Network (Faster R-CNN),” IEEE, Makassar, Indonesia, 2021.

[35] Y. Swapna, M. S. Reddy, J. V. Sai, N. S. S. Krishma and M. V. Teja, “DEEP LEARNING BASED ROAD TRAFFIC SIGN DETECTION AND,” Telangana, India, 2021.

[36] U. Kamal, T. I. Tonmoy, S. Das and K. Hasan, “Automatic Traffic Sign Detection and Recognition,” IEEE, Dhaka, Bangladesh, 2020.

[37] Cambridge in Colour, “Cambridge in Colour,” Cambridge in Colour, 2020. [Online]. Available: https://www.cambridgeincolour.com/tutorials/gamma-correction.htm. [Accessed 13 03 2022].

[38] S. H. Hsu and C. L. Huang, “Road sign detection and recognition using matching pursuit method,” 2001 Elsevier Science B.V., Hsinchu, 2000.

[39] L. Chunsheng and C. Faliang, “Fast and Robust Region of Interest Extraction for,” IEEE, Jinan, China, 2017.

[40] Arrive alive, “Traffic Signs of South Africa,” Arrive Alive, 2022. [Online]. Available: https://www.arrivealive.mobi/traffic-signs-of-south-africa. [Accessed 14 03 2022].

[41] R. Gandhi, “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,” Towards Data Science, 2018.