

LEARNING SIGN LANGUAGE USING SINGLE  
SHOT DETECTOR (SSD) AND MOBILENET

NIK AHMAD FARIHIN BIN MOHD ZULKIFLI

Bachelor of Computer Science  
(Software Engineering)

UNIVERSITI MALAYSIA PAHANG

## UNIVERSITI MALAYSIA PAHANG

### DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : NIK AHMAD FARIHIN BIN MOHD ZULKIFLI

Date of Birth

Title : LEARNING SIGN LANGUAGE USING SINGLE  
SHOT DETECTOR (SSD) AND MOBILENET

Academic Session : SEMESTER 2 2022/2023

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)\*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)\*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

\_\_\_\_\_  
(Student's Signature)

\_\_\_\_\_  
Date: 01/07/2023

\_\_\_\_\_  
(Supervisor's Signature)

DR. ZURIANI BINTI MUSTAFFA  
SENIOR LECTURER  
FACULTY OF COMPUTING  
UNIVERSITI MALAYSIA PAHANG  
26600 PEKAN PAHANG  
TEL: 09-4244734

\_\_\_\_\_  
Dr. Zuriani Binti Mustaffa  
Date: 17/07/2023



## **SUPERVISOR'S DECLARATION**

I hereby declare that I have checked this project and in my opinion, this project is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Science (Software Engineering) with Honors.

---

(Supervisor's Signature)

Full Name : Zuriani Mustafa

Position : Senior lecturer

Date : 17/07/2023



## **STUDENT'S DECLARATION**

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

---

(Student's Signature)

Full Name : NIK AHMAD FARIHIN BIN MOHD ZULKIFLI

ID Number : CB20179

Date : 01 July 2023

LEARNING SIGN LANGUAGE USING SINGLE SHOT DETECTOR (SSD) AND  
MOBILENET

NIK AHMAD FARIHIN BIN MOHD ZULKIFLI

Thesis submitted in fulfillment of the requirements  
for the award of the degree of  
Bachelor of Computer Science (Software Engineering) with Honors

Faculty of Computing  
UNIVERSITI MALAYSIA PAHANG

JULY 2023

## ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Zuriani Binti Mustaffa for her invaluable guidance, support, and encouragement throughout the course of this project. Their expertise and patience were instrumental in helping me navigate the research process and complete this project successfully.

My heartfelt gratitude goes to my family for their unwavering support and encouragement throughout this journey. They have been my rock, my source of strength and inspiration, and I couldn't have done it without them. Their love and understanding have been a constant source of motivation throughout my journey.

I would also like to express my sincere appreciation to my colleagues and friends for their support and encouragement throughout this project. Their contributions have been invaluable and have helped me to achieve my goals. They have been a constant source of inspiration, motivation, and support. They have always been there to lend a listening ear, share their wisdom and offer valuable suggestions. I am grateful for the camaraderie and friendship that we have shared.

Lastly, I extend my heartfelt thanks to all who have supported and assisted me in the completion of this project, particularly my thesis advisor, colleagues, friends, family, and every person involved. Your contributions and support have been invaluable and have played a vital role in the success of this project.

## **ABSTRAK**

Bahasa isyarat adalah bentuk komunikasi yang digunakan oleh komuniti yang pekak dan kurang pendengaran. Bahasa Isyarat Melayu (BIM) adalah bahasa isyarat rasmi yang diamalkan di Malaysia untuk berkomunikasi menggunakan isyarat tangan dan ekspresi wajah. Setiap isyarat dan kombinasinya mempunyai maksud yang berbeza, ini membuatkan ia lebih sukar bagi sesiapa untuk dengan mudahnya belajar Bahasa Isyarat Melayu. Oleh itu, kajian ini menghasilkan model pengesanan objek menggunakan Single Shot Detector (SSD) dan Mobilenet untuk mengesan Bahasa Isyarat dalam masa sebenar. Model ini hanya dilatih untuk mengesan isyarat statik yang tidak memerlukan kombinasi yang kompleks. Set data terdiri dari 2083 imej isyarat yang dikumpulkan dari laman web Kaggle dan dikumpulkan menggunakan kamera peribadi. Untuk fasa latihan, pengesanan dan pengujian, set data dibahagikan kepada nisbah 8:1:1. Dengan itu, tesis ini berjaya menghasilkan sistem yang sebenar dan tepat untuk pengenalan BIM menggunakan model SSD-Mobilenet, yang boleh memberi sumbangan kepada bidang pengenalan bahasa isyarat dan membantu untuk meningkatkan akses komunikasi untuk individu yang pekak dan kurang pendengaran.

## **ABSTRACT**

Sign languages are a form of communication used by the deaf and hard-of-hearing community. Malay Sign Language (MSL) is the official sign language that is practiced in Malaysia to communicate using hand signs and facial expressions. Every sign and its combination have a different meaning, this makes it quite hard for people to just casually pick up Malay Sign Language to learn. Therefore, this study presents an object detection model using Single Shot Detector (SSD) and Mobilenet to detect Sign Language in real time. This model is only trained to detect static signs which didn't require any complex combination. The dataset consists of 2000 sign images that were collected from a website called Kaggle and collected using a personal camera. For the training, validation, and testing phases, the dataset was divided into 8:1:1 respectively. In conclusion, this thesis has succeeded in developing a real-time and accurate system for MSL recognition using the SSD-Mobilenet model, which can contribute to the field of sign language recognition and help to improve communication access for deaf and hard-of-hearing individuals.



## TABLE OF CONTENT

<b>DECLARATION</b>	
<b>TITLE PAGE</b>	
<b>ACKNOWLEDGEMENTS</b>	<b>ii</b>
<b>ABSTRAK</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENT</b>	<b>v</b>
<b>LIST OF TABLES</b>	<b>viii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xi</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Introduction	1
1.2 Problem Statement	5
1.3 Objective	6
1.4 Scope	6
1.5 Thesis Organization	7
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>8</b>
2.1 Introduction	8
2.2 Existing Works	9
2.2.1 Region-based Convolutional Neural Network (R-CNN)	9
2.2.2 Faster Region-based Convolutional Neural Network (R-CNN)	10
2.2.3 You Only Look Once (YOLO)	12

2.3	Analysis/Comparison of Existing System	14
2.3.1	Table of Comparison	15
2.3.2	Analysis Review	16
2.3.3	Relevance of Comparison with project title	17
2.4	Summary	18
<b>CHAPTER 3 METHODOLOGY</b>		<b>19</b>
3.1	Introduction	19
3.2	Project Management Framework/Methodology	20
3.2.1	Data Capturing	21
3.2.2	Data Processing	22
3.2.3	Data Organization	24
3.2.4	Development	24
3.2.5	Model Training	25
3.2.6	Model Validation	26
3.2.7	Model Testing	26
3.2.8	Model Evaluation	26
3.2.9	Deploy Model to Mobile Apps	27
3.2.10	Real-time Testing	28
3.3	Project Requirement	28
3.4	Propose Design	30
3.5	Data Design	32
3.6	Proof of Initial Concept	33
3.7	Testing/Validation Plan	34
3.8	Potential Use of Proposed Solution	41
3.9	Gantt Chart	42

<b>CHAPTER 4 RESULTS AND DISCUSSION</b>	<b>43</b>
4.1 Introduction	43
4.2 Implementation Process	44
4.2.1 Data Description and Analysis	44
4.2.2 Data Labelling	52
4.2.3 Development of SSD-MobileNet model	54
4.2.4 Real-time testing	63
4.3 Result	66
4.3.1 Model evaluations	66
4.3.2 Model Testing	69
4.3.3 Real-Time testing	71
4.4 Discussion	76
<b>CHAPTER 5 CONCLUSION</b>	<b>77</b>
5.1 Introduction	77
5.2 Objective Revisited	77
5.3 Limitation	78
5.4 Future Work	79
<b>REFERENCES</b>	<b>ii</b>
<b>APPENDIX A</b>	<b>v</b>

## LIST OF TABLES

Table 2.1 Table of Comparison	15
Table 3.1 Samples of Datasets	21
Table 3.2 Software Items	29
Table 3.3 Hardware Items	29
Table 3.4 Samples of Datasets from Kaggle.com	32
Table 4.1 Sample of Datasets	44
Table 4.2 Model evaluation result	67
Table 4.3 Model testing result	70
Table 4.4 Real-time testing result	71
Table 4.5 Real-time Testing result	74
Table 4.6 Evaluation matrix result	74

## LIST OF FIGURES

Figure 1.1 Survey Chart 1	2
Figure 1.2 Survey Chart 2	2
Figure 1.3 Survey Chart 3	3
Figure 1.4 Survey Chart 4	3
Figure 1.5 Survey Chart 5	3
Figure 2.1 Selective Searching Process Figure	9
Figure 2.2 Convolutional Neural Network (CNN)	10
Figure 2.3 Faster R-CNN figure	11
Figure 2.4 Feature Maps	11
Figure 2.5 YOLO Grid divide process	12
Figure 2.6 Bounding Box Regression	13
Figure 2.7 Intersection over Union Formula	13
Figure 2.8 YOLO algorithm works	14
Figure 2.9 Graph of Accuracy and speed for All techniques	15
Figure 2.10 Accuracy detection for a different size	16
Figure 3.1 Flowchart of Model Development	20
Figure 3.2 Labelling using OpenLabeling software	22
Figure 3.3 XML File of labeled image	23
Figure 3.4 Line of label	23
Figure 3.5 Contents of the Images folder	24
Figure 3.6 Tensorboard example	25
Figure 3.7 Model Testing	26
Figure 3.8 Real-time testing	28
Figure 3.9 Single Shot Detector (SSD) MobileNet-v2 Architecture	30
Figure 3.10 Mobile Application Flowchart	31
Figure 3.11 Proof of Initial Concept	33
Figure 3.12 Malay Sign Language letters and numbers	34
Figure 3.13 Chosen Malay Words	34
Figure 4.1 Labelling using OpenLabeling	52
Figure 4.2 Annotations files	52
Figure 4.3 XML File of labeled image	53
Figure 4.4 Line of label	53
Figure 4.5 Connect Colab to Drive python code	54

Figure 4.6 Clone Object Detection API	54
Figure 4.7 Install Tensorflow	54
Figure 4.8 Create Training and testing record	55
Figure 4.9 Configure chosen Model SSD	55
Figure 4.10 Download SSD-MobileNet model file	56
Figure 4.11 Configure num_class	56
Figure 4.12 Configure file	57
Figure 4.13 Configure model learning rate	57
Figure 4.14 Load Tensorboard	58
Figure 4.15 Tensorboard interface	58
Figure 4.16 Train model command	58
Figure 4.17 Validation during training	59
Figure 4.18 Evaluation result	59
Figure 4.19 Stop model training	60
Figure 4.20 Export Inference Graph	60
Figure 4.21 Create tflite file	61
Figure 4.22 Model testing command	61
Figure 4.23 Testing model	62
Figure 4.24 Download TFlite file	62
Figure 4.25 Downloaded TFLite model	62
Figure 4.26 Tensorflow Mobile apps Github	63
Figure 4.27 Copy model into android asset	63
Figure 4.28 Change model name	64
Figure 4.29 Install mobile application	64
Figure 4.30 Real-time Testing in mobile apps	65
Figure 4.31 Total loss graph SSD-MobileNet model	66
Figure 4.32 Average Precision result after training	66
Figure 4.33 Average Recall result after training	66
Figure 4.34 Total loss result after training	66
Figure 4.35 Model Evaluation result after TFlite conversion	69

## LIST OF ABBREVIATIONS

MSL	Malay Sign Language
CNN	Convolutional Neural Network
SSD	Single Shot Detector
YOLO	You Only Look Once

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Introduction**

Deaf and mute are a type of disability that makes people unable to hear or speak to other people which makes it hard for them to communicate with others. In Malaysia alone, there are over 2 million people that have a hearing problem (Siti Hajar, 2019), and it is very common for people with a hearing problem to lose their ability to speak. People who are deaf or mute come in many ways, it could be genetics which means that they are born deaf or mute, and some people get it through injury or disease which led them to lose their ability to speak or hear voices. These people will be referred to as deaf-mute people from now on in this proposal.

To combat this communication barrier, researchers have developed many things to address this issue. One of the solutions is to use sign language to communicate with deaf-mute people. Sign languages are defined as languages that use our body parts to convey meaning to people without uttering any noise. It is first found in the 16<sup>th</sup> century when a Spanish Benedictine monk named Pedro Ponce de Leon create sign language for the hearing impaired at that time (DAYAS, 2019). Sign language has now developed since then and now and it has been used worldwide to communicate with deaf-mute people. There is also some slang in sign language that people use differently like how the Malay language has many slangs.

There are still a huge number of people who are unable to use sign language because they have no need for it or find it difficult to learn it, even though sign language is one of the most popular solutions to this communication problem. In any case, learning sign language is challenging because it is essentially a new language that we must master extensively before we can communicate using it. One of the common problems was finding it hard to study it by yourself by using books or videos from the internet because



we are unable to test and check if we are doing the sign correctly. This makes people that are new to learning this easily gave up learning it more.

A survey has been conducted for this project to study the knowledge and attitudes of Malaysia citizens towards Malay Sign Language. This survey covers three sections, the first section has the details of the respondents, second section ask questions about their attitudes and opinion towards Malay Sign Language, and third section has a simple test to test the respondents knowledge on MSL.

1. Have you ever learned Malay Sign Language?  
193 responses

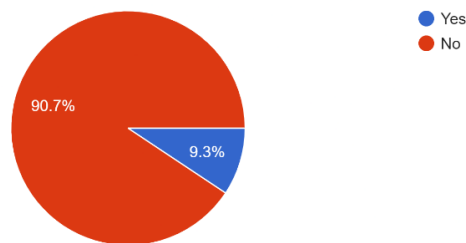


Figure 1.1 Survey Chart 1

2. Do you think its important to learn Malay Sign Language?  
193 responses

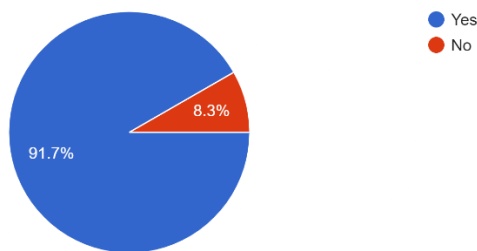


Figure 1.2 Survey Chart 2

From figure 1.1 and 1.2, it shows that over 90 percent of the respondents agree that it is important to learn Malay sign language but 90 percent of them have little to no experience learning Malay sign language.

3. Have you ever face a situation requiring to use Malay Sign Language?  
193 responses

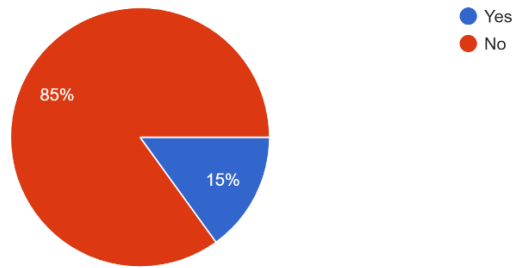


Figure 1.3 Survey Chart 3

4. Do you think its hard to learn Malay Sign Language without a tutor?  
193 responses

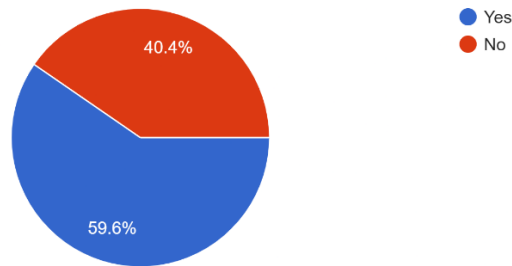


Figure 1.4 Survery Chart 4

5. Will learning Malay Sign Language helps you in daily life?  
193 responses

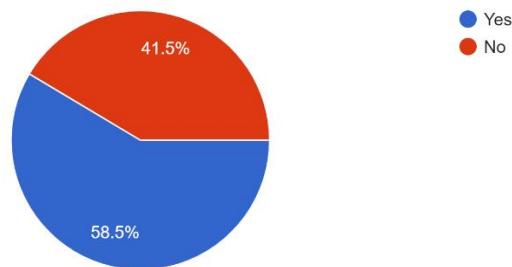


Figure 1.5 Survey Chart 5

From figure 1.3, it is quite apparent that many respondents rarely face a situation where they are required to use MSL. This is because most of the people nowadays will not be put into that situation unless they have some knowledge about MSL. Figure 1.5 shows that over 50 percent of the respondents agree that learning sign language will benefit them

in daily life. Figure 1.4 proves that learning sign language without any guidance or a tutor will be quite hard and challenging as 59 percent of the respondents agree with it. From this survey we could make an assumption that over 90 percent of Malaysia citizens wanted to learn MSL and interested in it, but most of the time, they find themselves in a situation where they do not want to get a tutor due to time and money constraints. This leads to them to learn MSL by themselves which ended up in disappointment because it's too hard to learn without a tutor.

Concerning the above matter, this project proposes a solution to use Single Shot Detectors and MobileNet to study sign language in Malaysia. Single Shot Detector with MobileNet is a deep learning-based technique that focused on embedded vision applications (Shafi et al., 2022). Deep Learning is a subset of Machine Learning, sign language model will be trained by using a Single Shot Detector for object detection and a MobileNet convolution neural network as the classifier (Pathak et al., 2018). A model is a file that is generated after going through training data to recognize patterns. The model will be used to make detection or decisions without being programmed manually to give the wanted output. There are many ways where Deep Learning is applied, such as image classification, face recognition, pose estimation, and many more. In this project, the Single Shot Detectors and MobileNet will be used to study sign language for object detection. Object detection means that the machine will learn how to recognize a certain pose and its meaning by learning from the training data that will be given.

In this study, the machines will be supplied with lots of image data of people doing Malay signs language that will be used for them to train and learn to recognize all the signs that have been selected for this project. This will create a model that can recognize Malay sign language from image input, the model will be used in a mobile phone application where user can use their camera to check if they are doing the sign correctly. Almost everyone in Malaysia has a smartphone nowadays, by deploying this model into a mobile application, users can use it everywhere and anytime that they want. This will help people to learn the Malay sign language easily by themselves.

## 1.2 Problem Statement

The current implementation of how people learn sign language is by going to classes either online or physically or they will find books or videos that they could use to learn sign language. The most effective way to learn sign language is to get a teacher that could teach step by step and will correct them when they make mistakes, they will also test them, and the student will be able to use them to communicate with the teacher. This method is effective, but it requires a lot of time and money spent for them to be able to learn and use sign language which they may or may never use, which deviates people to study by themselves either through books or internet content like videos and articles on the internet. The second method where people learn by themselves is suitable for people that don't have a lot of time and money on their side to spend on learning sign language.

The problem with learning by themselves is that they won't be able to know if they are doing the signs correctly. Without supervision from teachers, they will be going in blindly only by referring to anything they have on their hands and the internet, but they will be nobody for them to test and use what they learn. It makes it harder to learn and takes more time for them to be able to use sign language compared to people that learn from teachers or attend classes. Learning sign language wasn't that difficult but learning to be able to use it in daily life is difficult because it requires a lot of time until we can use it fluently (Jamie Berke, 2020).

At the current time and moment, there are already a lot of models that have been developed and trained to detect Malay Sign Language in a picture. In Malaysia alone, there are a lot of researchers that have developed this model but most of the method that has been used are not suitable for real-time object detection in an embedded device like mobile applications. Most of the time, the model is developed to be used in machines that run at a higher computation power like a computer. Other than that, researchers also developed a model that requires to use of external technology like Microsoft Kinect XBOX 360 camera or Microsoft Glove that can detect hand signs for sign language interpreters (Capilla, 2012).

Based on the highlighted issue, this project proposes a solution to use Single Shot Detectors and MobileNet to develop an object detection model that will learn sign language to help ease people's study of Malay sign language. In this proposal, we will

only use the formal Malay sign language also known as MSL, and only study the numbers, letters, and basic Malay Language for introductions which are “Nama”, “Saya”, and “Umur” (Darus et al., 2012). This model will be trained using hundreds and thousands of pictures of Malay sign language poses which will make the model able to recognize the signs from the poses that were given. This model is lightweight and can be deployed into an android phone application where people can test and check if they are doing the pose correctly or not by only posing in front of the camera and the model will check if the pose is similar to the one that has been learned and produce the confidence level and the sign meaning. In conclusion, this solution will help people to learn sign language easier without needing to find a teacher and they can do it faster by only using their phone.

### **1.3 Objective**

The objectives of this project are:

- i. To study the usage of object detection model using Single Shot Detector and MobileNet to study sign language.
- ii. To design and train the object detection model using Single Shot Detector and MobileNet to study Sign languages.
- iii. To evaluate the functionality of the developed object detection model to study sign language.

### **1.4 Scope**

User scope:

- i. Understand Malay Language
- ii. Age 7 years old and above.
- iii. Able to use a smartphone to study sign language means that the user can understand how to use a smartphone and use it as a platform to study.

System scope:

- i. Only used formal Malay sign language, this means any slang or short form in sign language is not covered in this project.

- ii. Only covered numbers from 0 to 9, all letters except for J and Z, and Malay words for basic introduction which are “Nama”, “saya” and “umur”, the words and letters that are chosen are static signs, which means that it doesn’t require any hand movement, meanwhile, those that are not covered are dynamic signs which require hand movement, this can’t be detected when using object detection.

Development scope:

- i. Develop environment using Python language in Google Colab for model development and Android Studio for object detection model deployment. The code for training and testing will be written in python language, meanwhile, the code for the mobile application will be written in Java language using the Android framework to deploy and test the trained model.
- ii. The dataset taken from Kaggle, and personal camera will be combined.
- iii. The dataset will be split into 80% for training, 10% for validation and 10% for testing.
- iv. Use OpenLabeling software for image annotations labeling.
- v. Use the Single Shot Detector and MobileNet-V2 Fpn lite Tensorflow pre-trained model to train MSL object detection model.
- vi. Deployed model to an android mobile application. When training and testing are done, the model will be downloaded and loaded into Android Studio to be deployed into a mobile application.
- vii. Use a desktop camera for collecting training and testing data and a phone camera for object detection in the mobile application.

## **1.5 Thesis Organization**

This thesis consists of five chapters. Chapter 1 shall discuss the introduction and the background of the study for this project. Chapter 2 will discuss the literature review where the proposed solution will be compared with other solutions that have been done. Chapter 3 will generally explain the methodology that will be used in this project to achieve the solution. Chapter 4 contains the implementation and results of testing from the solution that is produced. Chapter 5 will summarize the founding and conclude the results of this project.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

In this project, the pre-trained SSD-MobileNet-V2 fpn lite object detection model that will be used is developed by Tensorflow. Tensorflow is an open-source library for machine learning and Artificial Intelligence, by using functions that have been compiled into Tensorflow, people with almost little to no knowledge about how deep learning and machine learning work can create their deep learning model (Tensorflow, 2015).

In Deep Learning, there are many algorithms that we can use for object detection, there was a survey paper conducted on deep learning-based object detection, object detection algorithms for two-stage detectors like faster Region-based Convolutional Neural Network (RCNN), mask RCNN, Feature Pyramid Network (FPN), or one stage detectors You Only Look Once (YOLO), Single Shot Detector (SSD), RefineDet, and finally advanced detectors like CornerNet (Mittal et al., 2020). Other papers do compare in terms of the performance of the listed algorithms for object detection (Sharrab et al., 2021; Subbiah et al., 2020). In other papers, researchers also do a comparison of YOLO v3, Faster R-CNN, and SSD for real-time pill identification (Tan et al., 2021). There is also an article that talks about the difference between one-stage detectors and two-stage detectors to further our understanding of the two (Jason Brownlee, 2021)

In this project, Single Shot Detector MobileNet pre trained model will be used for our object detection model training, Single Shot Detector (SSD) MobileNet falls under the category of one stage detector for object detection algorithm. Single Shot Detector able to detect multiple boxes or objects in only one single shot which is considered quite fast and suitable for real-time object detection, SSD is explained in more detail and tested with many datasets in this cited paper (Liu et al., 2015). For this project, the SSD detection will be compared with R-CNN, Faster RCNN, and YOLO to test which

algorithm is the most suitable for sign language object detection in mobile phone applications.

## 2.2 Existing Works

### 2.2.1 Region-based Convolutional Neural Network (R-CNN)

Region-based Convolutional Neural Networks (R-CNN) is a type of machine learning model used in image processing and computer vision. The basic purpose of any R-CNN, which is specifically built for object detection, is to detect objects in any input image and define boundaries around them (Girshick et al., 2014). The way R-CNN works is divided into two parts as it is a two-staged detectors model, it divides the process of finding the region of interest (ROI) and classification processes using convolutional neural networks. First, when image input is given, it will do a selective search on the image (Sharif Elfouly, 2019).



Figure 2.1 Selective Searching Process Figure

From figure 2.1, the process of selective searching is done by creating numerous little windows or filters and then growing the region using the greedy algorithm. Then it finds the colors that are similar across the regions and combines them. Then it will use the formula below to calculate the similarity between regions:



$$S(a, b) = S_{\text{texture}}(a, b) + S_{\text{size}}(a, b) \quad 2.1$$

From formula 2.1, the  $S_{\text{texture}}(a, b)$  is visual similarity and  $S_{\text{size}}(a, b)$  is the similarity between the regions. The model keeps merging all the regions using this technique to increase the size of the regions. Once all the selection of regions has been created, CNN will take all the regions and create what's called a feature vector which is all the image regions in a much smaller dimension.

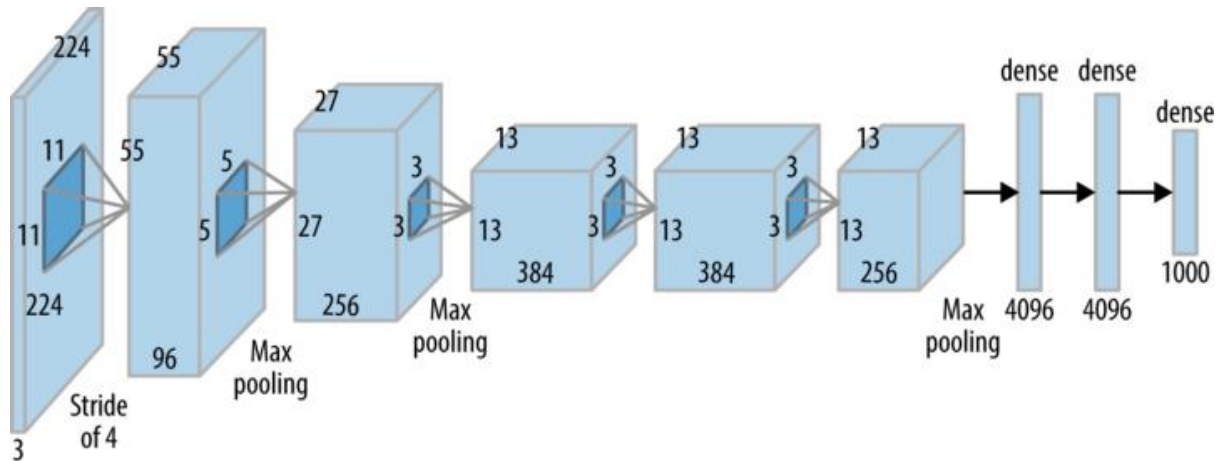


Figure 2.2 Convolutional Neural Network (CNN)

In figure 2.2, the images are extracted into smaller dimensions by using AlexNet as a feature extractor. The fact that the AlexNet receives the same input every time is another crucial consideration (227, 227, 3). However, the image suggestions have various shapes. Numerous of them are either smaller or larger than what is needed. We must therefore resize each region's proposals. Finally, it will use a Support Vector Machine (SVM) classifier to determine the class of the objects in the region.

### 2.2.2 Faster Region-based Convolutional Neural Network (R-CNN)

Faster R-CNN is the project of evolution from R-CNN to fast R-CNN to create Faster R-CNN, as the name stands it is the fastest of all of them (Ren et al., 2015). The main difference between Faster R-CNN and R-CNN is a fully convolutional deep network module that proposes regions. Faster R-CNN is separated into two parts, the first part is to propose regions by using a deep fully convolutional network known as Region Proposal Network (RPN), then the Fast-RCNN detector will use the proposed regions.

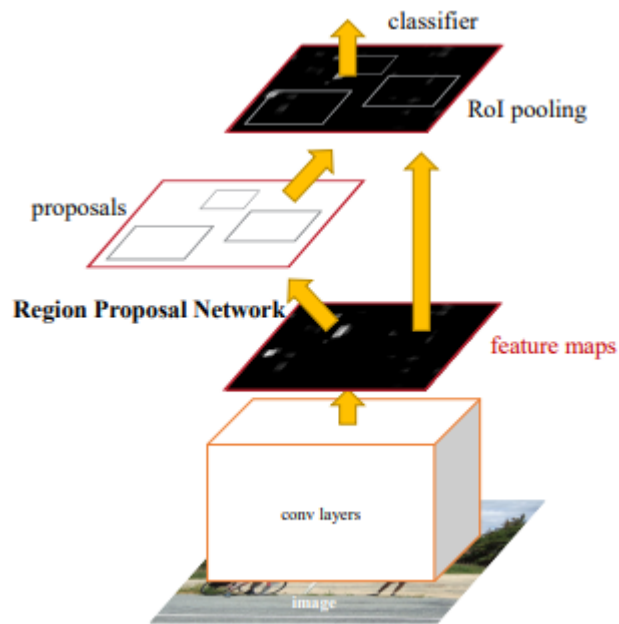


Figure 2.3 Faster R-CNN figure

In figure 2.3, the whole images function as an input into the convolutional layers where it will create feature maps by applying a filter to the image input. Feature maps are images that show patterns depending on the filter that is applied to the image. These papers show how the image is visualized when we applied a filter to them (Zeiler & Fergus, 2014).

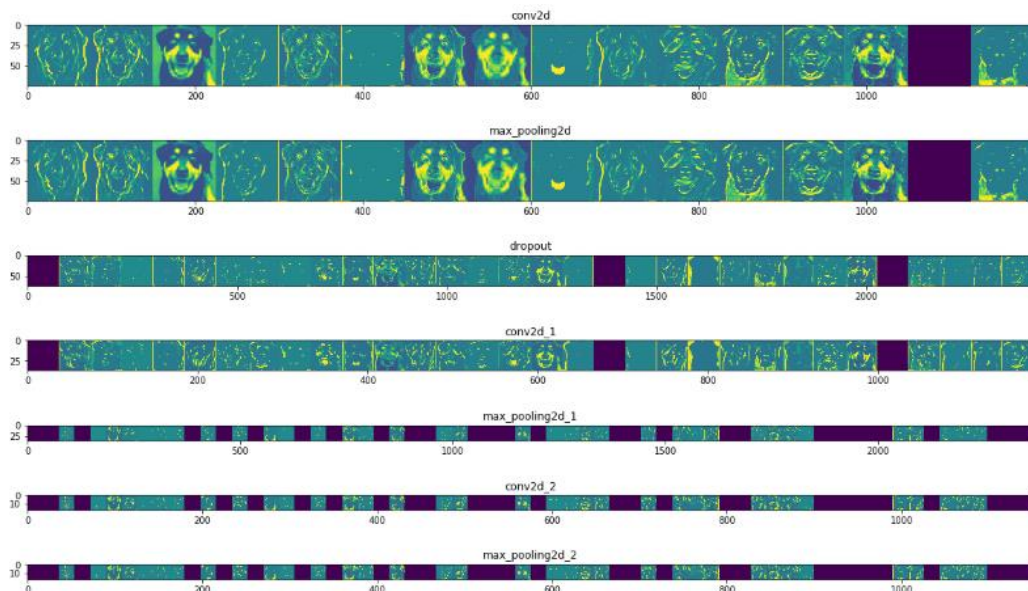


Figure 2.4 Feature Maps

Figure 2.4 shows the image output after going through the convolutional layer. By using these feature maps, it will be sent to RPN to create a region proposal. Region Proposal Network (RPN) will take feature maps input and give outputs of box and rectangle object proposals, each box will have its objectness score means the score of how much the box

likeliness to have an object in it. Finally, all of them will be gathered in the Region of Interest (ROI) pooling layer for the classification and identification process as how shown in figure 2.3.

### 2.2.3 You Only Look Once (YOLO)

You Only Look Once (YOLO) is a one-stage detector object detection technique, the first YOLO was released in 2015 introduced, until now there are several YOLO that has been released and developed (Bochkovskiy et al., 2020; Redmon et al., 2015). The current latest and newest official are YOLO v7 which many people consider as YOLO v5 because the previous v5 and v6 aren't been officially released (Hmrishav Bandyopadhyay, 2022). You Only Look Once as the name suggests means that it only looks at the image provided once and it will be able to detect the object available in the picture, this also enables it to do real-time object detection.

The YOLO algorithm divides the image into  $N$  grids, each of which has an equal-sized  $S \times S$  region. These  $N$  grids are each in charge of finding and locating the thing they contain. Accordingly, these grids forecast the object label, the likelihood that the object will be present in the cell, and  $B$  bounding box coordinates relative to their cell coordinates.

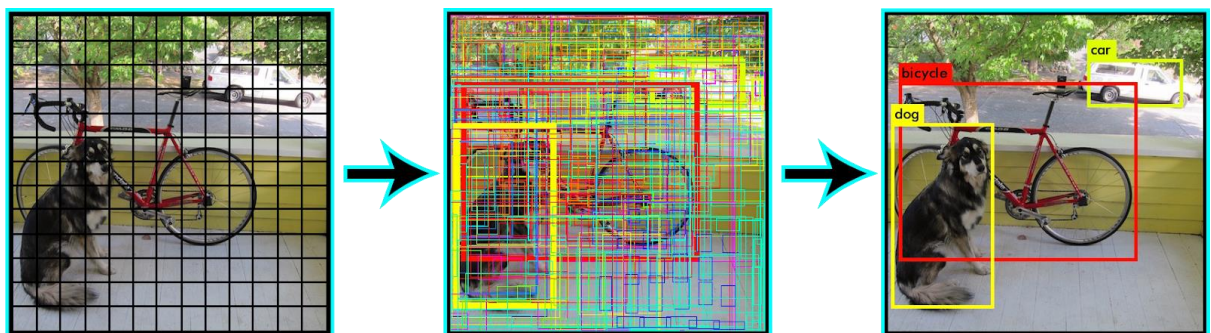


Figure 2.5 YOLO Grid divide process

In figure 2.5, the pictures are divided into several grids, then each grid will be used for bounding box regression, each bounding box will have the score if there are objects in it or not is represented as  $(P_c)$ , the height  $(B_h)$ , width  $(B_w)$ , probability class of the object  $(C)$ , and the bounding box center  $(B_x, B_y)$ .

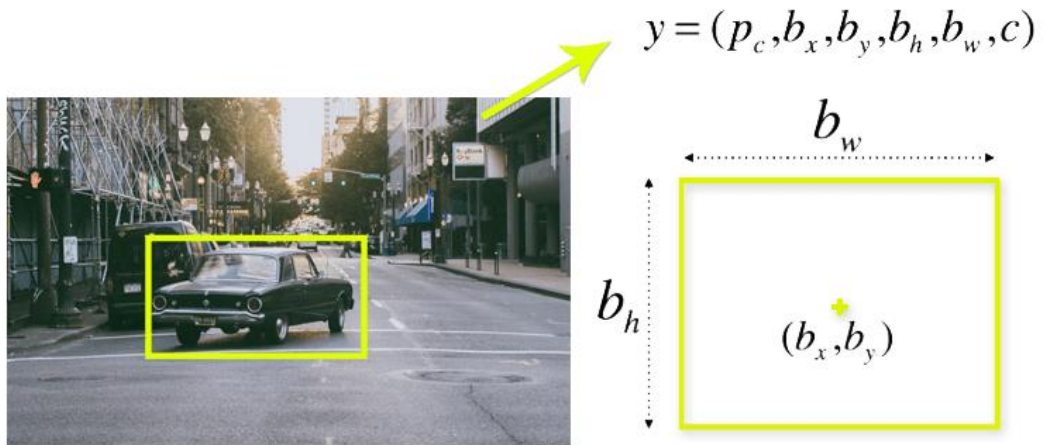


Figure 2.6 Bounding Box Regression

Figure 2.6 shows the bounding box with its attributes. YOLO also applied Intersection Over Union (IoU) formula. By using this formula, it will calculate how much each box intersects with the other.

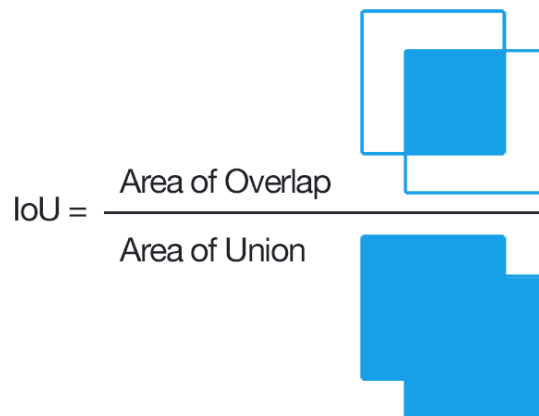


Figure 2.7 Intersection over Union Formula

Figure 2.7 shows the IoU formula where the intersection of the bounding box will be divided by the whole union of the bounding box. This formula will be used to create the output box that surrounds the object perfectly because if IoU equals one means that the predicted box and the output box are perfectly aligned.

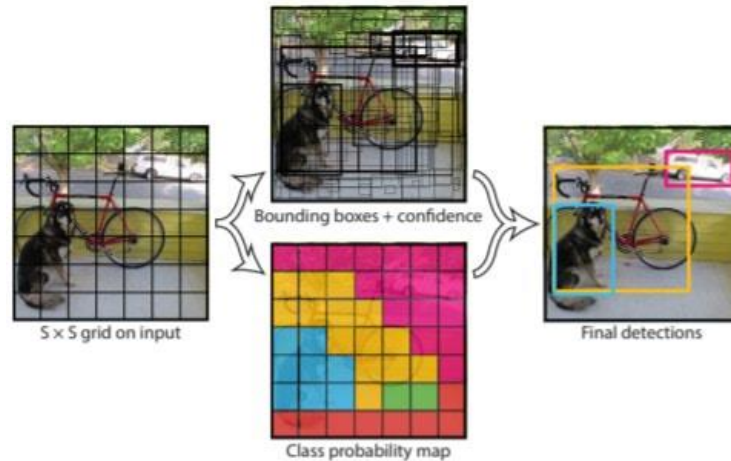


Figure 2.8 YOLO algorithm works

Figure 2.8 shows the ways the YOLO algorithm works in object detection. Firstly, the image is first separated into grid cells. The B bounding boxes are predicted in each grid cell, along with confidence ratings. To determine the class of each object, the cells forecast the class probability. For instance, a bicycle, a dog, and a car are examples of at least three different types of objects. A single convolutional neural network is used to make all the detections concurrently. The projected bounding boxes are equal to the actual boxes of the objects thanks to the intersection over the union. This phenomenon gets rid of bounding boxes that aren't necessary or don't match the properties of the objects (like height and width). The final detection will be made up of special bounding boxes that precisely suit the objects (Grace Karimi, 2021).

### 2.3 Analysis/Comparison of Existing System

In this part, a comparison of Single Shot Detector (SSD) with Region-based Convolutional Neural Network (R-CNN), Faster R-CNN, and YOLO will be done by doing a table of comparison for between each algorithm in object detection.

### 2.3.1 Table of Comparison

The following table shows the comparison between three existing systems R-CNN, Faster R-CNN, YOLO, and the chosen technique for this project SSD.

Table 2.1 Table of Comparison

Criteria	R-CNN	Faster R-CNN	YOLO	SSD (Chosen Technique)
Type of Detectors	Two Stage Detectors	Two Stage Detectors	One Stage Detectors	One Stage Detectors
Region Proposal Method	Selective search	Region proposal network	Residual blocks, Bounding box regression, Intersection over Union (IoU)	Multi-box Detectors
Detection Timing	40-50 sec	0.2 seconds	Real-time	Real-time
Computation time	High	Low	Low	Low
Mean Average Precision on Pascal VOC 2007 test dataset (%)	58.5%	73.2%	78.6%	81.6%

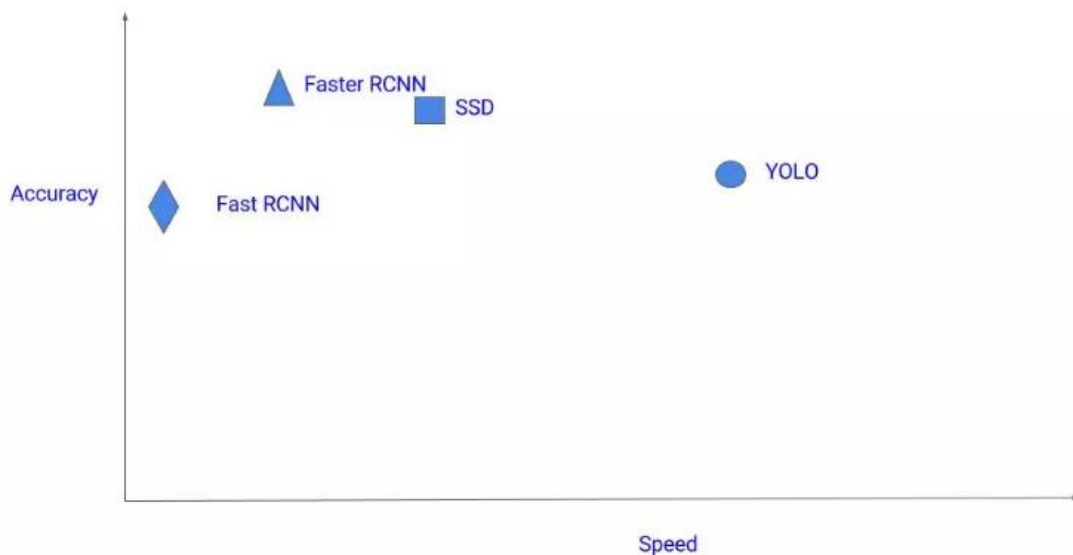


Figure 2.9 Graph of Accuracy and speed for All techniques

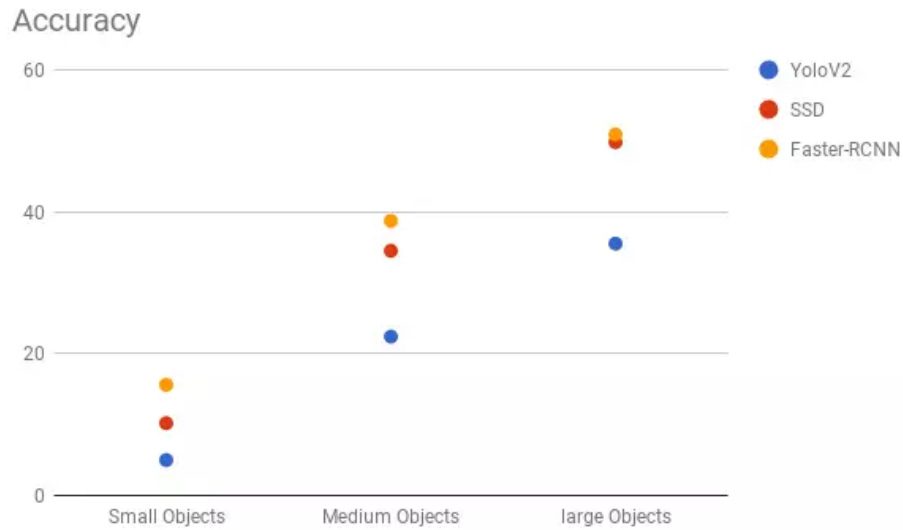


Figure 2.10 Accuracy detection for a different size

All data from table 2.1 figures 2.9 and 2.10 are taken from these references (PapersWithCode, 2022) (Ankit Sachan, 2017).

### 2.3.2 Analysis Review

From table 2.1, we could see that the only algorithm that couldn't keep up is R-CNN since it is the oldest among all of them. Region Convolutional Neural Network is a two-stage detector, method that is used to create the region proposal is Selective searching. Selective search requires a lot of time which increases the computation time from low to high. It also produces a low mean average precision when tested with the Pascal Visual Object Classes test dataset 2007, it measly scored a 58.5% accuracy. Thus, R-CNN could not be used for this project where the final output will be used on a mobile phone.

Next, Faster R-CNN is the improved version of its R-CNN predecessor and is also a two staged detectors similar to R-CNN. Faster R-CNN used Regional Proposal Network (RPN) to create a proposal network for the bounding box of the objects, this is a faster method compared to Selective searching in R-CNN. This reduced the computation time by a lot and reduce the detection time from 40-50 seconds to 0.2 seconds. It also scored quite a high mean average precision on the PASCAL VOC test dataset 2007 which is 73.2%. Figure 2.9 shows that Faster R-CNN is not as fast as YOLO and SDD, but it got



the highest accuracy score due to its slower speed. In figure 2.10, Faster R-CNN got the best accuracy in all different sizes from small to large objects.

Next, YOLO is another algorithm for object detection that is a one-stage detector similar to SSD. YOLO used residual blocks to create the bounding box for its objects, this method can produce a box at a high speed, and this enables YOLO to be used in real-time with a low computation time. YOLO v2 scores quite high mean average precision on the PASCAL VOC test dataset 2007 which are 78.6%. From figure 2.9, YOLO can be seen to be the fastest but got the lowest accuracy compared to Faster R-CNN and SSD. This is also shown in figure 2.10 where YOLO v2 got the worse accuracy in all different sizes from small to large objects.

Finally, SSD is a one-stage detector's deep learning technique. It could run the detection in real time, which means that the detection and detection results are instant. The only downside of SSD is that it falls a little bit short in terms of speed and a little bit lower in accuracy compared to faster R-CNN as shown in figure 2.9. This also means that the SSD is the most balance between the two since it still values accuracy over YOLO speed and SSD is faster compared to R-CNN in terms of detection time. Similarly in figure 2.10, SSD always falls between Faster R-CNN and YOLO in all different sizes. Thus it is safe to conclude that SSD is a good choice for object detection techniques in this project.

### **2.3.3 Relevance of Comparison with project title**

This comparison of the existing system will help to further understand how to choose the correct technique for deep learning in object detection. R-CNN, Faster R-CNN, YOLO, and SSD are good techniques for object detection, but all of them have their advantages and disadvantages. The chosen Single Shot Detector (SSD) has been proven to be a good technique after doing a comparison with other object detection algorithms, this also means that SSD could be applied when doing sign language study using object detection.



## **2.4 Summary**

In summary, this chapter has discussed the algorithm techniques that can be used for deep learning object detection techniques. All the functionality and advantages of SSD will be used to solve sign language study problems. In the next chapter, this report will cover the methodology that will be used for this project.

## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Introduction**

This chapter will describe the methodology that will be chosen to develop the model for the Malay Sign Language study using Single Shot Detector (SDD) and MobileNet. Every process that will be done will be described in detail, the project requirements, proposed design, data design, and proof of initial concept will be provided in this chapter. Then, the testing and validation plan will be constructed to test the functionality of the object detection model in learning sign language. Next, the potential use of the proposed solution will be discussed and finally, the Gantt chart planning of the whole project will be provided.

### 3.2 Project Management Framework/Methodology

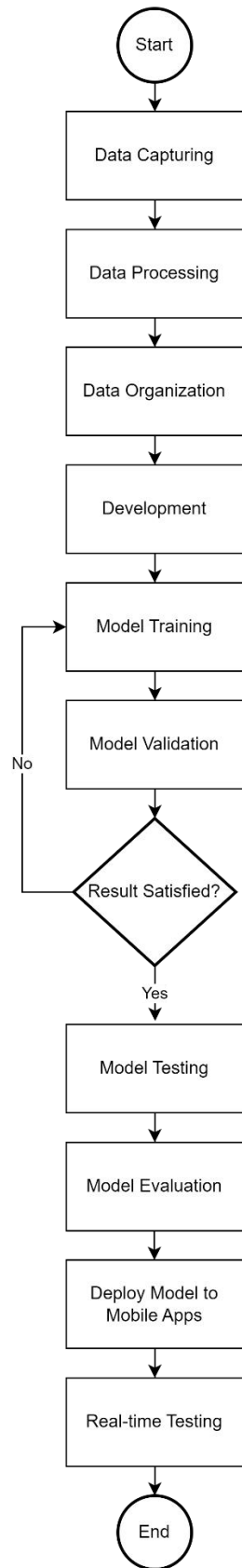


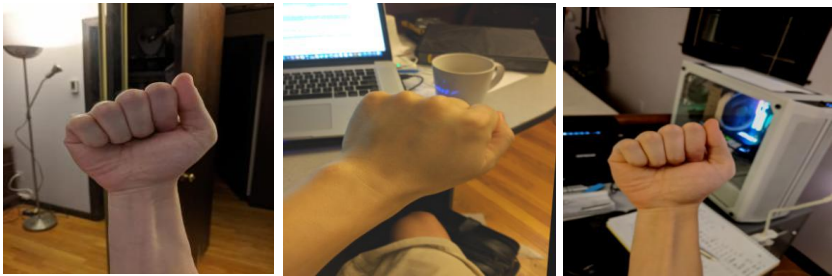
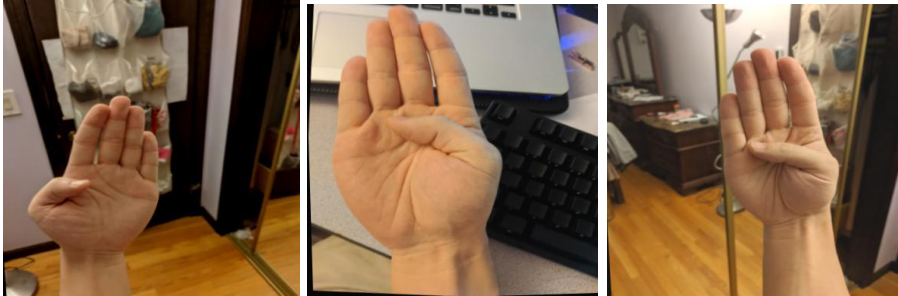
Figure 3.1 Flowchart of Model Development

Figure 3.1 shows the flowchart of all processes that will be done to develop the model for Malay sign language object detection. In this project, nine steps need to be done to achieve the target of this project. The nine steps are data capturing, data processing, data organization, development, model training, model validation, model testing, deployment model to mobile applications, and real-time testing.

### 3.2.1 Data Capturing

During the data capture process, a portion of the image dataset will be obtained from the Kaggle website, which provides a variety of datasets in different fields that can be utilized for machine learning and model training. For this project, the dataset will consist of signs that are similar to the English Sign language, including letters and 27 signs commonly used in Malay sign language. To facilitate data collection, the dataset for letters will be obtained from Kaggle, as referenced (Mavi & Dikle, 2022). In addition to this, a desktop camera attached to a personal computer and a personal phone camera will be used to capture images of numbers, letters, and selected Malay words which are "Nama," "saya," and "umur." For each specific sign, approximately fifty to hundreds of images will be collected, resulting in a dataset with images of each sign. It's worth noting that a large dataset is crucial for effective model training, as insufficient data can lead to overfitting.

Table 3.1 Samples of Datasets

Label	Signs
A	
B	

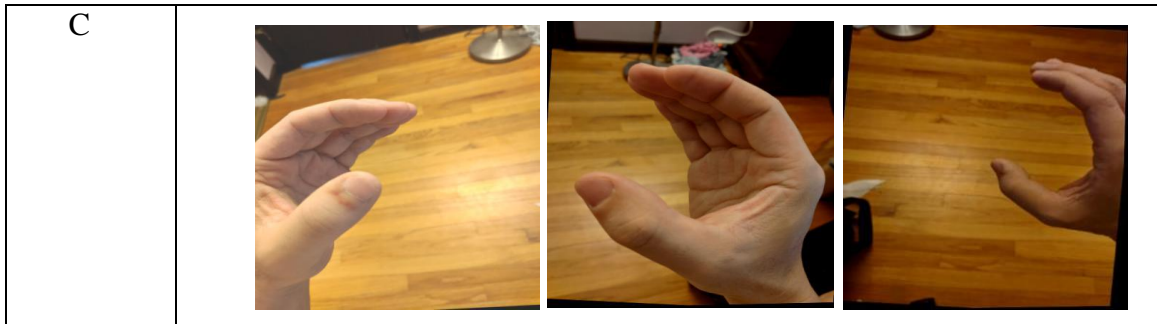


Table 3.1 shows the samples of datasets, where it has signs of A, B, and C and pictures for each sign.

### 3.2.2 Data Processing

During the data processing stage, the images that have been collected will be labeled using OpenLabeling software. This software is an open-source image annotation tool that enables image labeling by drawing visual boxes around objects in images to identify and label them (João Cartucho, 2018).

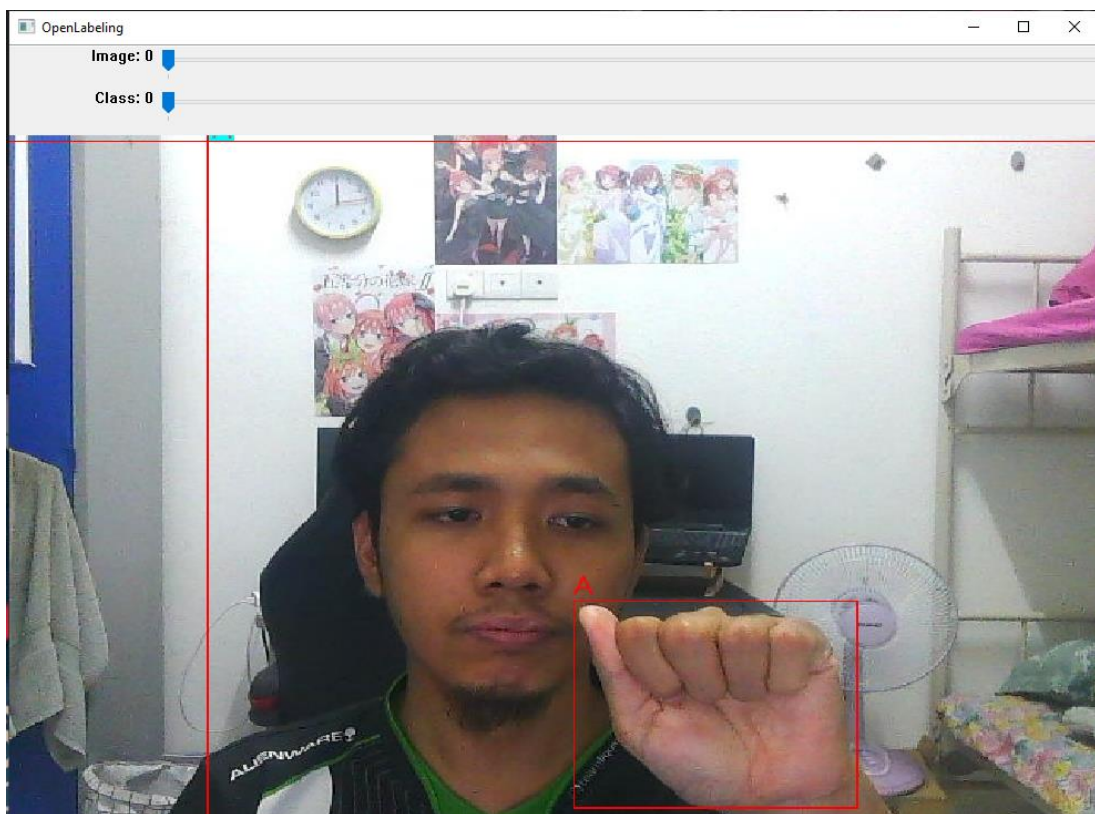


Figure 3.2 Labelling using OpenLabeling software

Figure 3.2 shows the process of image labeling using OpenLabeling software. For this project, OpenLabeling will be used to draw visual boxes around signs inside the images that have been collected, this will create an XML file of the labeled images.

```
annotation>
  <folder>collectedimages</folder>
  <filename>A.b5c4c267-57a4-11ed-ac87-f09e4a80946c.jpg</filename>
  <path>D:\RealTimeObjectDetection\Tensorflow\workspace\images\collectedimages\A.b5c4c267-57a4-11ed-ac87-f09e4a80946c.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>640</width>
    <height>480</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>A</name>
    <pose>Unspecified</pose>
    <truncated>1</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1</xmin>
      <ymin>327</ymin>
      <xmax>147</xmax>
      <ymax>480</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 3.3 XML File of labeled image

Figure 3.3 shows the XML file that is created from Labeling software which contains the image of the signs with its label.

```
<name>A</name>
```

Figure 3.4 Line of label

Figure 3.4 shows that its name is 'A', which means that the sign is labeled as 'A'. All the images will be labeled using OpenLabeling software in the data processing phase.

### 3.2.3 Data Organization

In data organization, all the images and the XML files will be separated into two folders called Images and annotations. In Images, all the images without a label will be stored meanwhile annotations file will store the XML files which contains all the label for the images.

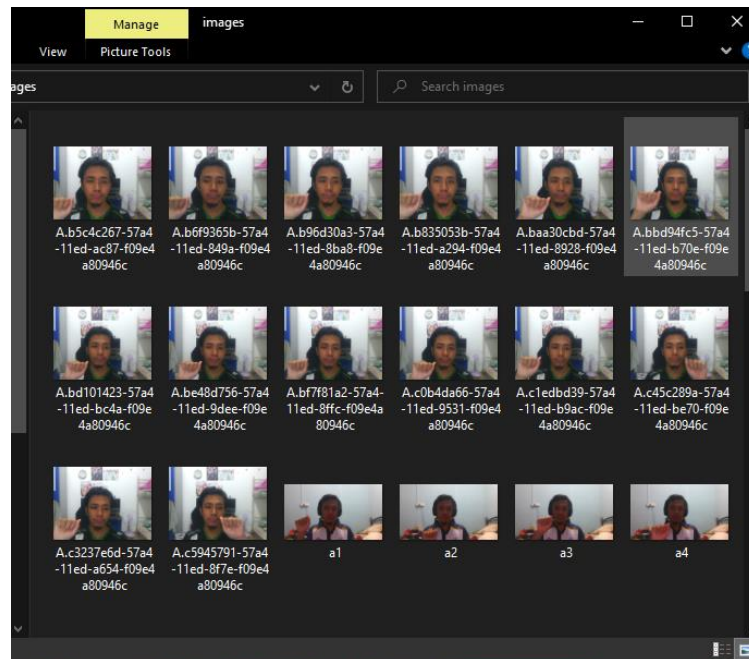


Figure 3.5 Contents of the Images folder

Then, both the images and the annotations from Figures 3.5 will be compressed and uploaded into Google Drive to ease data training during model training and testing. This will store data in cloud storage so that will act as a backup if anything happens to the local storage.

### 3.2.4 Development

Development of the model will be done in Google Colab notebooks where the codes will be developed using python language. All the processes such as preparing the environment, dataset retrieval from Google Drive, training data, testing data, and converting the model into a Tensorflow lite model will be done in Google Colab. Google Colab is a cloud-based notebook that allows people to write and run their python codes without using their computer processor because it used cloud computing that allows it to run in the server that is provided by Google.

### 3.2.5 Model Training

During the model training phase, the dataset will be split into training, validation and testing data with a ratio of 8:1:1. The Malay sign language detection model will be trained using Single Shot Detector (SSD) and MobileNet pre-trained model provided by the Tensorflow detection model. Configuration settings such as the number of classes, batch size, and the number of steps for training the detector will be done. Using Tensorflow pre-trained models offers an advantage where a checkpoint is created every 1000 steps of training, allowing the training to be resumed from the checkpoint if it gets interrupted. Additionally, Tensorflow provides a Tensorboard that displays the current training progress, including the learning rate, classification loss, localization loss, regularization loss, and the total loss. The training will be stopped once it reaches 50,000 steps as specified in the configuration file. If the training results do not meet the desired loss of 0.1, the training will be repeated with different batch sizes until the desired results are achieved.

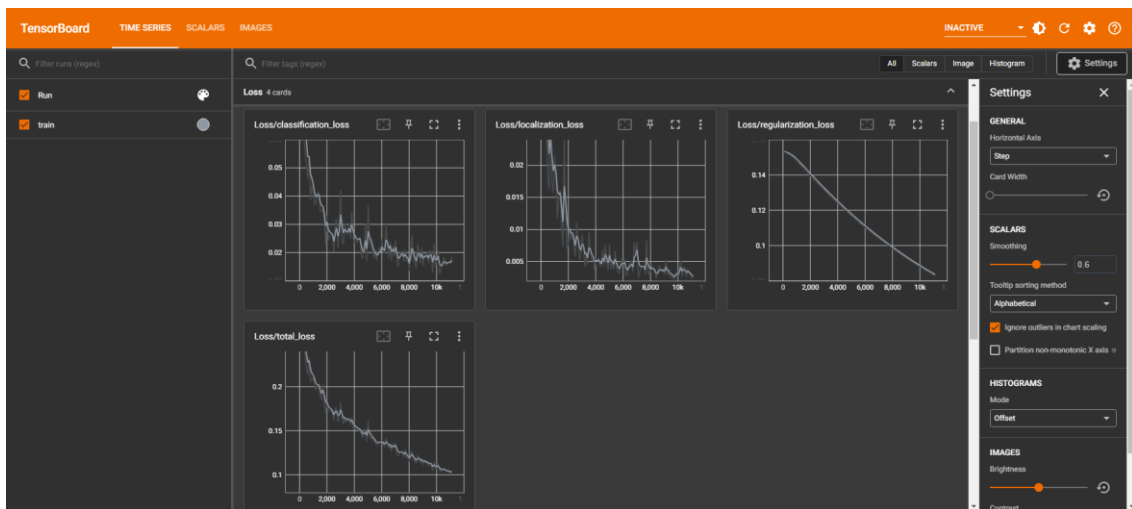


Figure 3.6 Tensorboard example

Figure 3.6 shows the example of Tensorboard that is provided by Tensorflow, the graph will be updated from time to time during training depending on the batch size.



### 3.2.6 Model Validation

Once the training of the model is completed, it is essential to validate the model's performance to ensure that it meets the desired level of accuracy. Python commands provided by TensorFlow can be utilized to perform the validation of the model by using the prepared validation data. The validation process involves calculating the average precision and recall of the developed model. If the average precision and recall score surpasses 0.8, it signifies that the model is capable of detecting and classify the signs accurately within the validation data.

### 3.2.7 Model Testing

After the validation, the model will be tested in Google Colab, as it was trained there. The prepared test images will be loaded into the model for testing purposes.

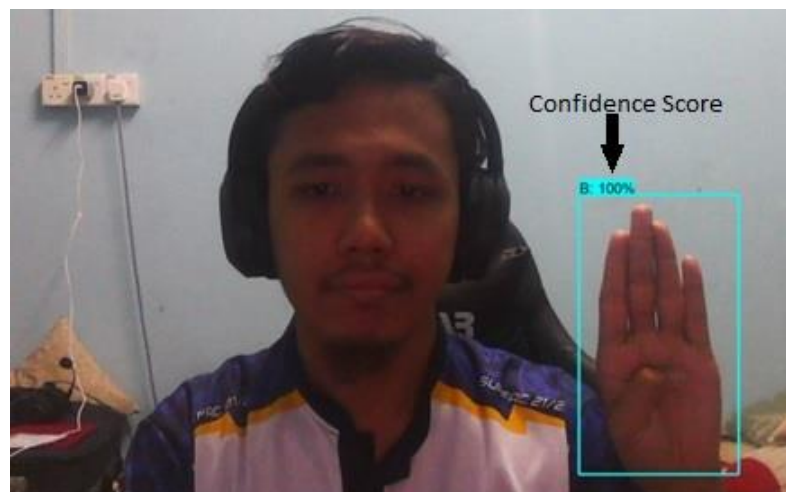


Figure 3.7 Model Testing

An example of the model's performance is shown in Figure 3.7, where it successfully identified the Malay sign language object labeled as 'B' with a 100% confidence score. The model drew boxes around the object and provided a confidence score for each sign. All prepared images will be utilized to evaluate the model's performance.

### 3.2.8 Model Evaluation

The 'accuracy' metric is commonly used to evaluate the performance of object detection models. However, in this project, in addition to accuracy, precision, recall, and F1-score will also be calculated to evaluate the model's performance (Das et al., 2023). To calculate these metrics, true positive (TP), true negative (TN), false positive (FP), and

false negative (FN) variables are required. A true positive occurs when the model correctly classifies a test example for class A as class A, while a false negative occurs when the model incorrectly classifies a test example of class A as some other class B. A false positive occurs when a test example from class A is incorrectly classifies as some other class B, whereas a true negative occurs when the model correctly classifies that a test object from some other class B is not a test example from class A. The formulas for calculating all these metrics are provided below, and they will be utilized to evaluate the model's performance in this project:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FN + FP + TN)} \quad (3.1)$$

$$\text{Precision} = \frac{TP}{(TP + FP)} \quad (3.2)$$

$$\text{Recall} = \frac{TP}{(TP + FN)} \quad (3.3)$$

$$\text{F1 - Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \quad (3.4)$$

### 3.2.9 Deploy Model to Mobile Apps

In this phase, the model that has been trained and tested needs to be converted from a Tensorflow model (TF) to a Tensorflow Lite model to make it suitable for mobile applications, embedded systems, and Internet of Things devices, as Tensorflow Lite is optimized to run with less computing power. Once the conversion is done, the Tensorflow Lite model will be downloaded and integrated into an Android application developed in Android Studio. To simplify the process, Tensorflow provides its library and fully developed mobile application code, and the downloaded model will be used to replace the existing model in the mobile application.

### 3.2.10 Real-time Testing

After replacing the model, the mobile application can be installed on a virtual device or an android phone through android studio. Once installed, the phone camera can be used to capture a sign language pose, and the model will detect the sign and determine its meaning.

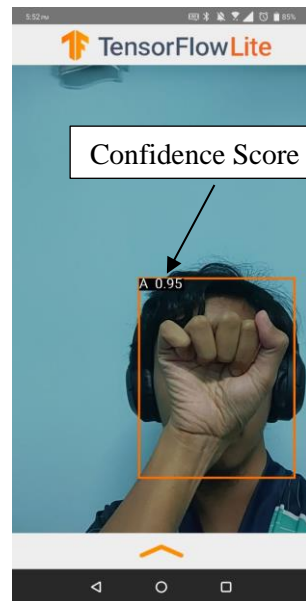


Figure 3.8 Real-time testing

Figure 3.8 shows an example of real-time testing using the Malay sign language object detection model. The figure shows the box around the sign with the 'A' label and the confidence score of 0.95 for the sign.

### 3.3 Project Requirement

Sign languages are not an easy language that people could pick up and learn instantly, it requires a lot of time to learn and correct teaching. Most people would appreciate learning it by themselves without needing to hire a personal teacher or join a class because they want to learn it without hindering their daily lives. In this project, the model that will be created will help people to learn or test their sign language knowledge. This model should be able to detect Malay sign languages that are posed in front of their phone camera and give the meaning and confidence score of it.

Developing this model requires a lot of Malay sign language image data, this data will be required to train the model to study sign language. With all the data available, the model for sign language object detection will be trained to learn Malay sign language and produce a model that can detect and classify the signs inside images. The model will be trained using a Single Shot Detector MobileNet pre-trained model developed by Tensorflow for object detection.

This project focuses on one type of sign language, which is static signs, meaning it only covers signs without any movement. Specifically, it covers the letters and numbers in Malay sign language, as well as a few Malay words such as "Nama", "saya", and "umur". It is important to note that this model does not include complex combinations of sign language or non-manual gestures. Additionally, this model will only be available on Android phones since the mobile application code is developed using the Android Studio IDE.

This part will list all the software that will be used to develop the Malay Sign Language object detection model:

Table 3.2 Software Items

<b>Software</b>	<b>Purpose</b>
Microsoft Word 360	To create and compile all documents.
OpenLabeling	Image processing and labeling.
Google browser	To code in Google Colab notebooks for the development of the model.
Android Studio	To deploy object detection model and testing.

Table 3.3 Hardware Items

<b>Hardware</b>	<b>Purpose</b>
Personal Desktop Computer	To do all the processes related to the project.
Desktop Camera	To collect images for the dataset.
Android ASUS ROG phone	To install the mobile application for model testing.

### 3.4 Propose Design

In this project, the algorithm that will be used is Single Shot Detector (SSD) MobileNet-V2 pre-trained model. This pre-trained model will be used to train our model for Malay Sign Language object detection where it will use the architecture of SSD MobileNet.

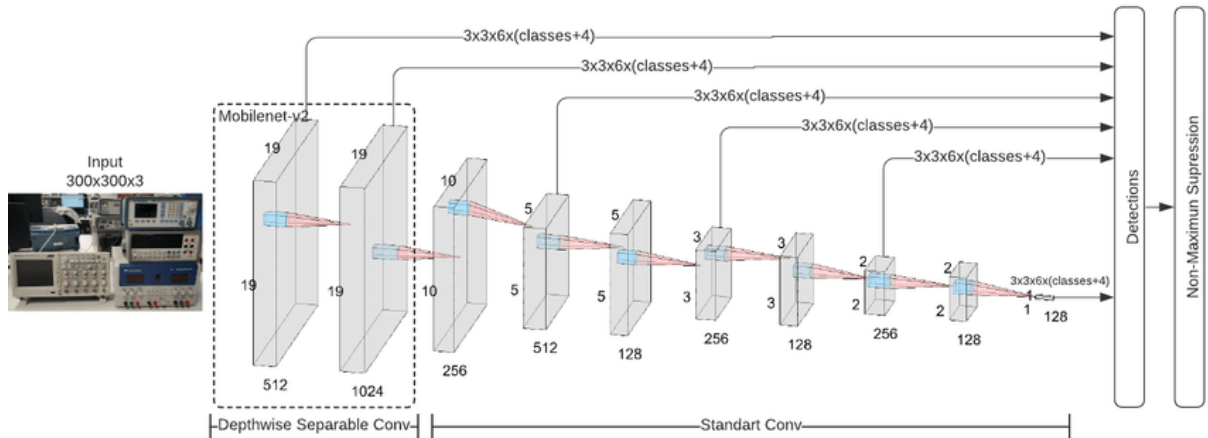


Figure 3.9 Single Shot Detector (SSD) MobileNet-v2 Architecture

Figure 3.9 shows the Single Shot Detector (SSD) MobileNet-v2 architecture, this deep convolutional neural network will be used to train our sign language detection model (Estrada et al., 2022). In this project, MobileNet-V2 will be used to train and classified images of Malay Sign Language. MobileNet-V2 is an inverted residual blocks with a linear bottlenecks (Sandler et al., 2018). The arrangement of the blocks will follow a **thin-thick-thick-thin** pattern. Point wise convolutional layers will be utilized to produce a higher dimensional feature map. The Rectified Linear Unit (ReLU) will be used as the activation function for the feature map to eliminate any values below 0, which would lead to data loss. To reduce data loss, a higher dimension feature map will be created. Depth wise convolutional layers with ReLU activation will then be applied to the features map. Finally, a point wise convolutional layer will be implemented to the feature maps to create a smaller dimension feature map. To prevent information loss, a linear activation function will be used for the final activation function, hence the term "linear bottlenecks".

The next step in this project is to utilize the Single Shot Detector (SSD), also known as the Single Shot Multibox Detector, which is capable of detecting multiple items in a single shot. The SSD approach employs a feed-forward convolutional network that

produces a fixed-size set of bounding boxes and scores for object class instances within those boxes. In this method, MobileNet-v2 is utilized as the backbone detection network for multi-scale object recognition, providing a variety of feature maps with varying dimensions. SSD applies a convolutional layer with small convolutional filters to give confidence scores and class offsets for a fixed set of standard bounding boxes. The non-maximum suppression technique, which uses Intersection over Union, is subsequently used to eliminate boxes with low probabilities of containing an object. MobileNet-SSDv2 will be utilized in this project due to its development specifically for mobile applications and embedded devices with limited computing power while still achieving high classification accuracy.

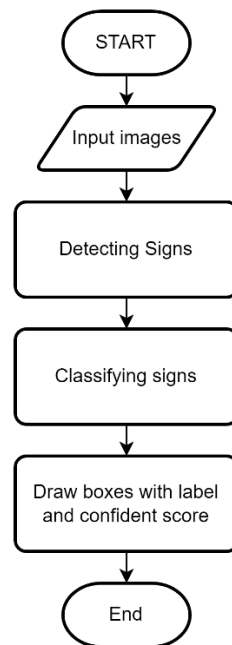


Figure 3.10 Mobile Application Flowchart

Figure 3.10 shows the flow of the mobile application with the Malay sign language object detection model. The flowchart starts with the user giving input images, input images here are live stream camera input which means that the user only poses in front of their camera. Next, the mobile apps will search if there is any sign language available in the picture, if there are sign images, it will classify it then draw boxes around it and show the confidence score of it.

### 3.5 Data Design

Table 3.4 Samples of Datasets from Kaggle.com

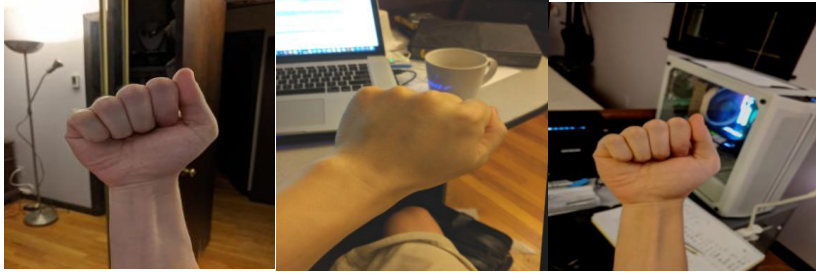
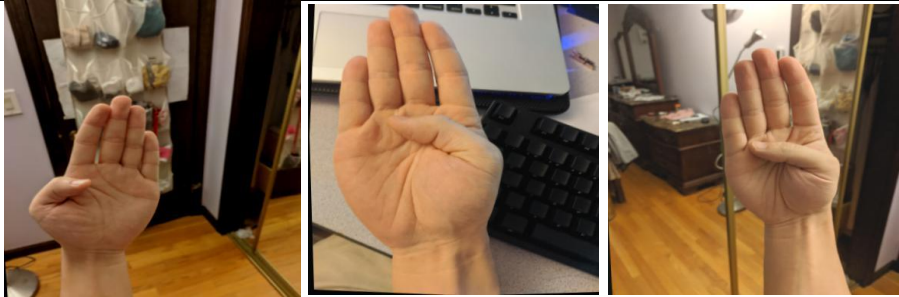

Label	Signs
A	
B	
C	

Table 3.4 shows the sample of dataset that will be used for the letters of the sign language. American sign language letters dataset that is available on Kaggle, this dataset consists of images files for all the sign language letters except letters J and Z because those two are dynamic signs which require motions (Ammar Alhaj Ali, 2020). This dataset has been created for the purpose of studying sign language using machine learning and object detection. Since Malay letters for sign language are similar to American sign language, this dataset could be used to ease the process of data capturing. Next, all other data images for the number signs will be created by capturing .jpg images using a camera and labeling them using OpenLabeling software.

### 3.6 Proof of Initial Concept

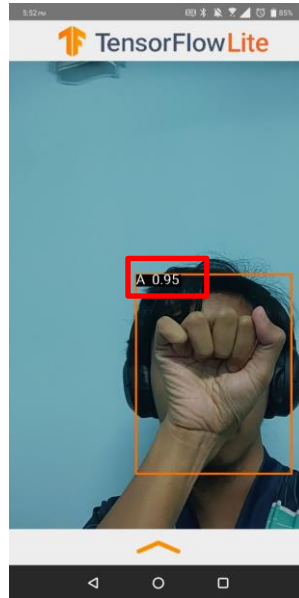


Figure 3.11 Proof of Initial Concept

The initial concept for the Malay sign language object detection model using Single Shot Detector MobileNet-v2 in an android mobile application is demonstrated in Figure 3.11. This model was developed with a focus on detecting two classes of data, namely the letters 'A' and 'B', indicating that the model has been trained to identify 'A' and 'B' signs in images. As depicted in Figure 3.11, the model successfully detects the 'A' sign in the image and draws bounding boxes around it with a confidence score of 0.95 along with its label. Other than that, this method of using SSD-MobileNet V2 has been done before in a paper where they develop a real-time sign language recognition for Macedonian Sign language (Kralevska et al., 2022).



### 3.7 Testing/Validation Plan

To test the functionality of the model, the user can use signs language that has been trained which are the letters and number from one to nine and chosen Malay words, if the output given is the same as the sign that was shown, it means that the model has been successfully trained to detect and classify Malay Sign Language. The result of the test will be recorded in the test plan document below. Figure 3.12 and 3.13 shows the signs that have been chosen for the model that will be developed respectively.

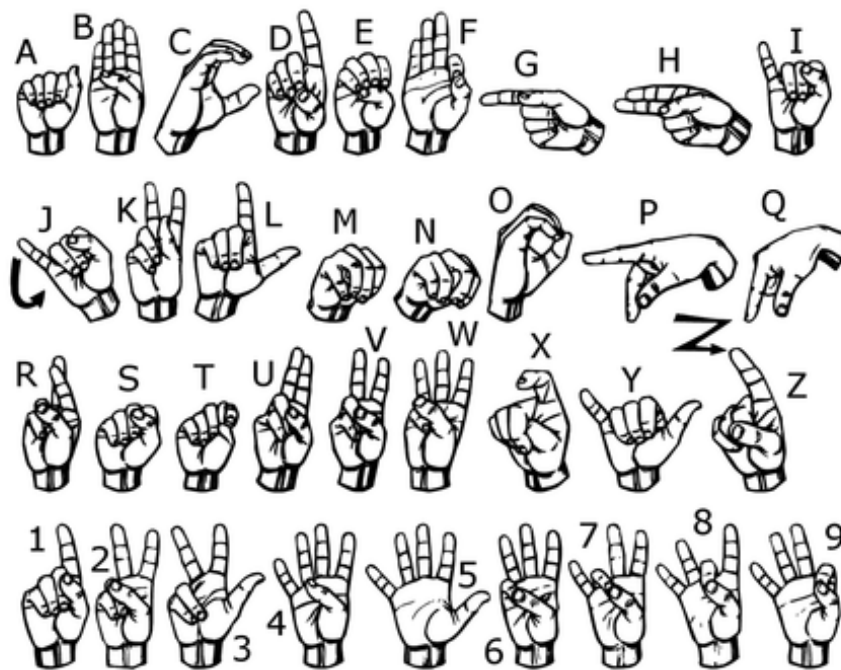


Figure 3.12 Malay Sign Language letters and numbers

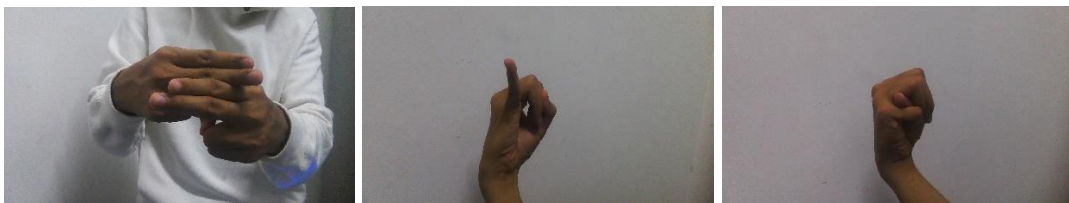







Figure 3.13 Chosen Malay Words








**Test Plan Document**








Name : \_\_\_\_\_









Signature : \_\_\_\_\_







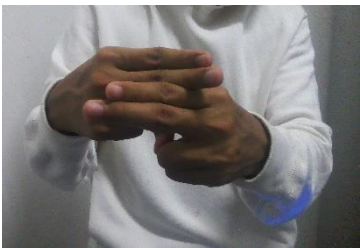
Date : \_\_\_\_\_



No.	Sign Image	Expected Signs label	Output label	Output Confidence score
1	 a	A		
2	 b	B		
3	 c	C		
4	 d	D		
5	 e	E		

6	 f	F		
7	 g	G		
8	 h	H		
9	 i	I		
10	 k	K		
11	 l	L		
12	 m	M		

13	 n	N		
14	 o	O		
15	 p	P		
16	 q	Q		
17	 r	R		
18	 s	S		
19	 t	T		

20	 <b>U</b>	U		
21	 <b>V</b>	V		
22	 <b>W</b>	W		
23	 <b>X</b>	X		
24	 <b>Y</b>	Y		
25	 <b>1</b>	1		
26	 <b>2</b>	2		
27	 <b>3</b>	3		

28	 4	4		
29	 5	5		
30	 6	6		
31	 7	7		
32	 8	8		
33	 9	9		
34		Nama		

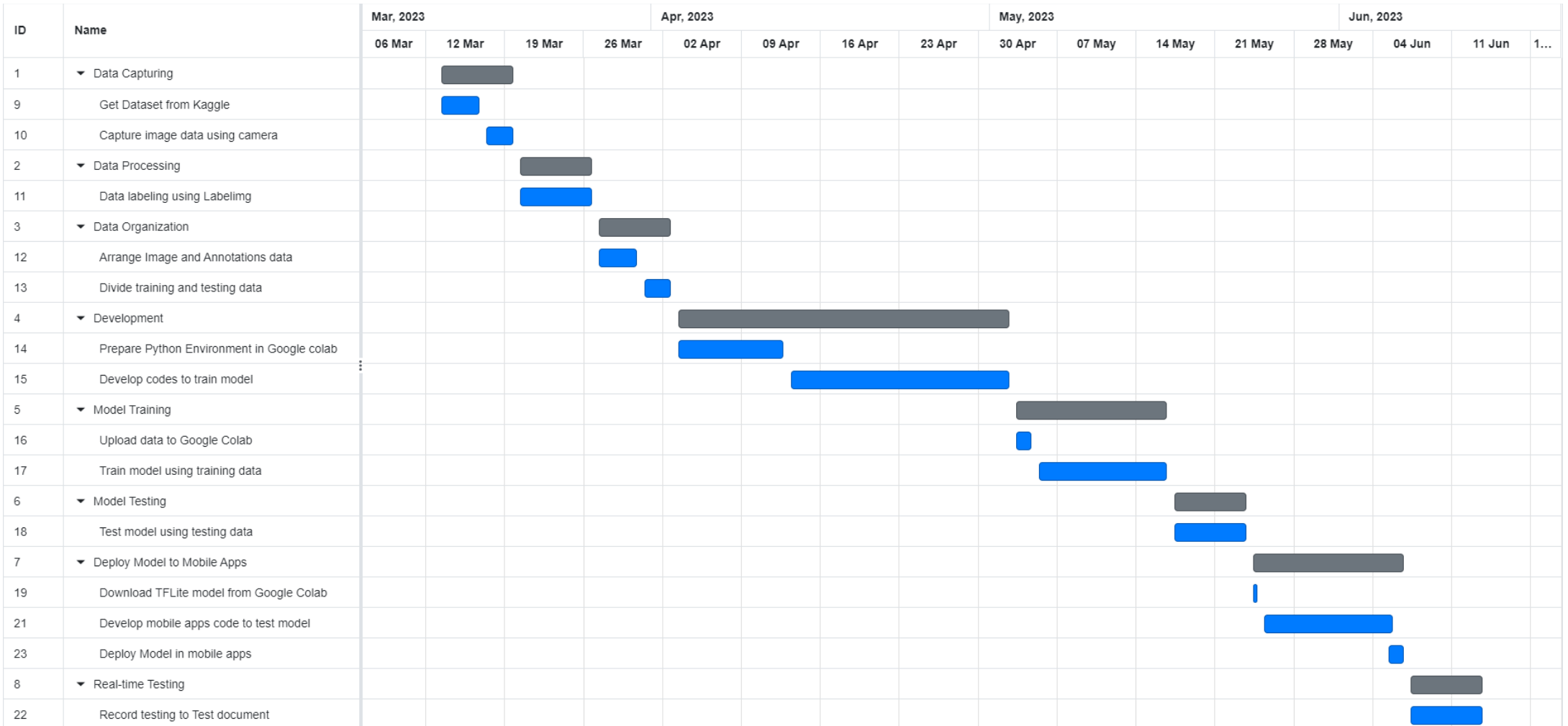
35	 <p>A hand gesture where the index finger is extended upwards, and the other fingers are curled into the palm. A small letter 'S' is visible at the bottom left of the image.</p>	Saya		
36	 <p>A hand gesture where the index and middle fingers are extended upwards, and the other fingers are curled into the palm.</p>	Umur		

### **3.8 Potential Use of Proposed Solution**

Although this proposed solution cannot fully facilitate communication for people who rely on sign language in daily life as it only detects signs as objects within images and not gestures or facial expressions, it can still be a useful tool for beginners in the learning process. The model is particularly helpful for individuals who are learning letters, numbers, and basic introductions to Malay Sign Language, but may not have access to a teacher. By deploying this model as an android mobile application, it can serve as a helpful starting point for those who wish to learn sign language. Furthermore, this application can help raise awareness about the importance of sign language in Malaysia, thereby increasing people's knowledge and appreciation of Malay sign language.



### 3.9 Gantt Chart



## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

Chapter 4 discusses the development of a real-time object detection model for Malay Sign Language using Single Shot Detector (SSD) MobileNet. The model is developed using transfer learning, a method that involves using an existing model to solve another problem in machine (Dianne Castillo, 2021). For this project, the pre-trained Single Shot Detector-MobileNet object detection model developed by TensorFlow is trained to detect images of Malay Sign Language.

The training process of the SSD-MobileNet model using the Python programming language is explained in detail. Python is a flexible programming language that can import many libraries and extensions used in machine learning. The TensorFlow library is used to train this object detection model as it provides functionality that facilitates the training and monitoring process of the model. The chapter also presents and discusses the accuracy and testing of the developed model.

The evaluation, model testing, and real-time testing have yielded conclusive results regarding the suitability of the SSD-MobileNet model for sign language detection in images. Through comparisons of the models' capabilities in detecting signs, it is evident that the SSD-MobileNet model outperformed the others. Following closely behind is the SSD-Resnet50 model, while the SSD-Efficientdet-d0 model lagged significantly behind. The SSD-MobileNet model showcased remarkable performance after 20,000 training steps, consistently achieving over 90% accuracy in all conducted tests.



## 4.2 Implementation Process






As explained in the previous chapter, the goal of implementing the Single Shot Detector-MobileNet object detection model is to develop a model that can detect Malay Sign Language in real-time video capture. The development environment for training the pre-trained SSD-MobileNet model is created using Python programming language in Google Colab Notebook.

### 4.2.1 Data Description and Analysis






The datasets that have been collected using desktop camera and Kaggle contain 2083 images of numbers, letters and basic introduction for Malay Sign Language, the dataset have been split into 8:1:1 ratio for training, validation, and testing which means 1666 images for training, 208 images for validation and 209 images for testing. Table 4.1 shows the sample of the collected dataset.




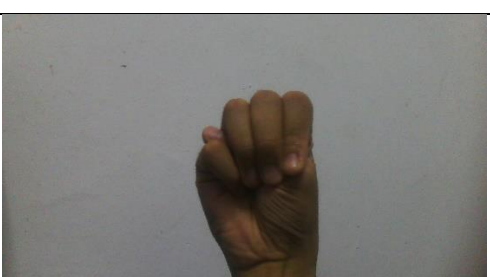

Table 4.1 Sample of Datasets






No.	Signs	Labels	Quantity
1		1	50
2		2	50

3	 A hand gesture showing three fingers extended: the index, middle, and ring fingers. The thumb and pinky are curled in towards the palm.	3	50
4	 A hand gesture showing four fingers extended: the index, middle, ring, and pinky fingers. The thumb is curled in towards the palm.	4	50
5	 A hand gesture showing all five fingers extended in a fan shape.	5	50
6	 A hand gesture showing six fingers extended: the index, middle, ring, pinky, and thumb. The thumb is extended outwards.	6	50
7	 A hand gesture showing seven fingers extended: the index, middle, ring, pinky, thumb, and the other index finger. The thumb and the other index finger are extended outwards.	7	50






8		8	50
9		9	50
10		A	62
11		B	60
12		C	60





13		D	60
14		E	60
15		F	60
16		G	60
17		H	60

18		I	60
19		K	60
20		L	60
21		M	60
22		N	60

23		O	60
24		P	60
25		Q	60
26		R	60
27		S	60



28		T	60
29		U	60
30		V	60
31		W	60
32		X	60

33		Y	60
34		Nama	84
35		Saya	50
36		Umur	57

## 4.2.2 Data Labelling

To train SSD-MobileNet model, the image datasets need to be labelled so the model will know the parts that it needs to focus on. In this project, the images have been labelled using Software OpenLabeling.

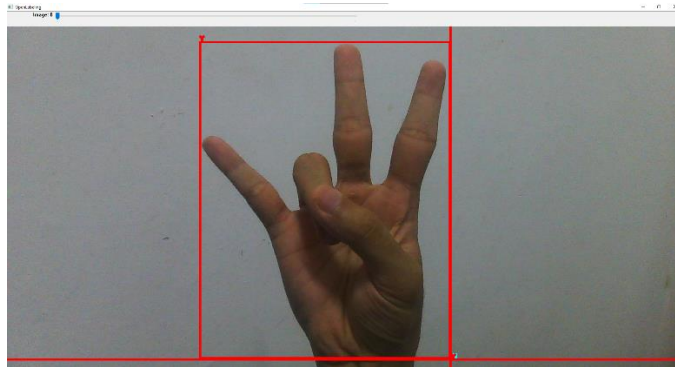
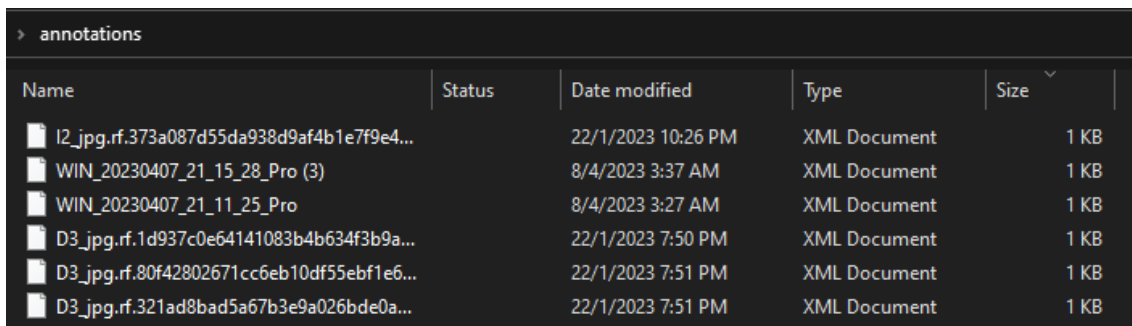


Figure 4.1 Labelling using OpenLabeling

For this project, the images were labeled using OpenLabeling software, which involved drawing visual boxes around the signs within the collected images. This process creates an XML file of the labeled images, as shown in Figure 4.1.



Name	Status	Date modified	Type	Size
I2_jpg.rf.373a087d55da938d9af4b1e7f9e4...		22/1/2023 10:26 PM	XML Document	1 KB
WIN_20230407_21_15_28_Pro (3)		8/4/2023 3:37 AM	XML Document	1 KB
WIN_20230407_21_11_25_Pro		8/4/2023 3:27 AM	XML Document	1 KB
D3_jpg.rf.1d937c0e64141083b4b634f3b9a...		22/1/2023 7:50 PM	XML Document	1 KB
D3_jpg.rf.80f42802671cc6eb10df55ebf1e6...		22/1/2023 7:51 PM	XML Document	1 KB
D3_jpg.rf.321ad8bad5a67b3e9a026bde0a...		22/1/2023 7:51 PM	XML Document	1 KB

Figure 4.2 Annotations files

Figure 4.2 shows the collections of annotations that were created using OpenLabeling software. Each image in the dataset has been annotated with visual boxes that indicate the location of signs in the image. These annotations were saved as corresponding XML files and will be used to train the SSD-MobileNet model.

```

annotation>
  <folder>collectedimages</folder>
  <filename>A.b5c4c267-57a4-11ed-ac87-f09e4a80946c.jpg</filename>
  <path>D:\RealTimeObjectDetection\Tensorflow\workspace\images\collecte
dimages\A.b5c4c267-57a4-11ed-ac87-f09e4a80946c.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>640</width>
    <height>480</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>A</name>
    <pose>Unspecified</pose>
    <truncated>1</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1</xmin>
      <ymin>327</ymin>
      <xmax>147</xmax>
      <ymax>480</ymax>
    </bndbox>
  </object>
</annotation>

```

Figure 4.3 XML File of labeled image

Figure 4.3 shows the XML file contents that is created from OpenLabeling software which contains the image of the signs with its label.

```
<name>A</name>
```

Figure 4.4 Line of label

Figure 4.4 shows that its name is 'A', which means that the sign is labeled as 'A'. All the images will be labeled using OpenLabeling software in the data processing phase.

### 4.2.3 Development of SSD-MobileNet model

First, the dataset is uploaded and stored in Google Drive to prevent any problems and act as backup from the local drive. The dataset will be loaded into the python Google Colab notebook by connecting it to Google Drive. Figure 4.5 shows the codes to connect Google Drive to Google Colab.

```
from google.colab import drive
drive.mount('/content/gdrive')

# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is equal to /mydrive
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
```

Figure 4.5 Connect Colab to Drive python code

```
!git clone --depth 1 https://github.com/tensorflow/models
```

Figure 4.6 Clone Object Detection API

After connecting Google Drive to Google Colab, the next step is to clone the Tensorflow models repository from Github in order to use the object detection API. Figure 4.6 displays the Python codes required to clone the SSD-MobileNet model from Tensorflow object detection API and import it into the Colab Virtual Machine. Utilizing this API and its function is necessary for developing any Python libraries for object detection model development. These lines of code will install all necessary libraries and verify their compatibility and functionality for this model development.

```
!pip install tensorflow==2.8.0
```

Figure 4.7 Install Tensorflow

To proceed, Tensorflow needs to be installed in the Google Colab environment. This installation will provide access to the Tensorflow libraries and functions necessary for object detection model training. Figure 4.7 depicts the Python code utilized to install Tensorflow 2.8 in Google Colab.

```

!python3 create_tfrecord.py
--csv_input=/mydrive/customtf/train_labels.csv
--labelmap=labelmap.txt
--image_dir=/mydrive/customtf/train
--output_path=train.tfrecord

!python3 create_tfrecord.py
--csv_input=/mydrive/customtf/validation_labels.csv
--labelmap=labelmap.txt
--image_dir=/mydrive/customtf/validation
--output_path=val.tfrecord

```

Figure 4.8 Create Training and testing record

In object detection training, the training and testing record are files that contain information about the annotated images used to train and test the model. The training record is a file that contains a list of images with their corresponding annotations, which is used to train the model. The testing record, on the other hand, contains a separate list of images with their annotations that are not used in training, but rather for testing the accuracy and performance of the trained model. Displayed in Figure 4.8 are the Python codes to generate the training and testing records using the datasets that have been split into training and testing. As highlighted in the red box of Figure 4.8, the generated files are labeled as "train.tfrecord" for training and "val.tfrecord" for testing the model.

```

chosen_model = 'ssd-mobilenet-v2-fpn-lite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2-fpn-lite-320': {
        'model_name': 'ssd_mobilenet_v2_fpn-lite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpn-lite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpn-lite_320x320_coco17_tpu-8.tar.gz',
    },
}

model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']

```

Figure 4.9 Configure chosen Model SSD

The Python code displayed in Figure 4.9 sets the SSD-Mobilenet model as the chosen model for object detection training. The subsequent line of code specifies the configuration file name for the SSD-Mobilenet model.

```

# Download pre-trained model weights
import tarfile
download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' +
pretrained_checkpoint
!wget {download_tar}
tar = tarfile.open(pretrained_checkpoint)
tar.extractall()
tar.close()

# Download training configuration file for model
download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/o
bject_detection/configs/tf2/' + base pipeline file
!wget {download_config}

```

Figure 4.10 Download SSD-MobileNet model file

The Python code in Figure 4.10 demonstrates the process of downloading the configuration files required for the SSD-MobileNet model. As can be seen in the figure, the filename that will be downloaded is specified in the last line of the code, using the "!wget" function.

```

model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 36
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
  }
}

```

Figure 4.11 Configure num\_class

Once the configuration files have been downloaded, they must be modified to suit the dataset. The initial modification involves changing the number of classes. As indicated in the figure, the number of classes has been adjusted to 36, which corresponds to the number of signs that will be used for object detection training.

```
train_config: {
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint: "/mydrive/customtf/models/mymodel/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0"
  fine_tune_checkpoint_type: "detection"
  batch_size: 16
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 40000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}
```

Figure 4.12 Configure file

In object detection model training, the batch size and number of steps are important parameters to configure. The batch size refers to the number of images that the model processes in each iteration of the training process. Larger batch sizes can lead to faster training times but require more memory. Smaller batch sizes can slow down training but require less memory. The number of steps, on the other hand, refers to the total number of iterations that the training process runs for. Each step involves processing one batch of images through the model. The number of steps required depends on the model's complexity and the dataset's size. Both batch size and number of steps are hyperparameters that need to be tuned for optimal training and best results. Figure 4.12 shows that the batch size is set to 16, and the number of steps is set to 40,000.

```
optimizer {
  momentum_optimizer: {
    learning_rate: {
      cosine_decay_learning_rate {
        learning_rate_base: .08
        total_steps: 50000
        warmup_learning_rate: .026666
        warmup_steps: 1000
      }
    }
  }
}
```

Figure 4.13 Configure model learning rate

Additionally, in order to achieve optimal results, the learning rate has been set to 0.08 as depicted in figure 4.13. The learning rate is a hyperparameter that determines the step size at which the model weights are adjusted during the training process. A higher learning rate can result in faster convergence but may also cause the model to overshoot the optimal weights, resulting in poorer performance. Conversely, a lower learning rate can slow down the training process but may result in a more accurate model. Therefore, choosing the right learning rate is crucial for achieving the best results in object detection model training.



```
#load tensorboard
%load_ext tensorboard
%tensorboard --logdir '/content/gdrive/MyDrive/customTF2/training'
```

Figure 4.14 Load Tensorboard

Figure 4.14 shows the line of code to load Tensorboard in a Google Colab Notebook.

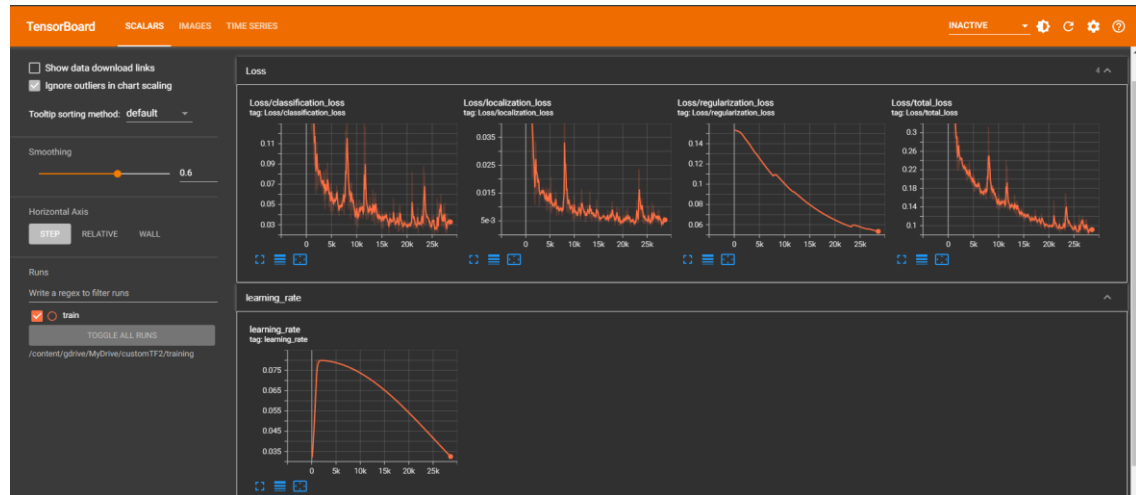


Figure 4.15 Tensorboard interface

In Figure 4.15, the Tensorboard interface is displayed during the training of the SSD-MobileNet model. Tensorboard is a visualization tool that comes with Tensorflow to help monitor the progress of the training process. It provides graphs and metrics that allow you to visualize the performance of the model over time, including accuracy and loss values. The precision and recall graphs for the mean average precision of the model are also displayed in the Tensorboard interface.

```
# Run training!
!python /content/models/research/object_detection/model_main_tf2.py \
  --pipeline_config_path={pipeline_file} \
  --model_dir={model_dir} \
  --alsologtostderr \
  --num_train_steps={num_steps} \
  --sample_1_of_n_eval_examples=1
```

Figure 4.16 Train model command

Figure 4.16 displays the command to initiate the training of the ssd-mobilenet model using the "model\_main\_tf2.py" script file. This script file is responsible for training the SSD-MobileNet model based on the configuration set in the configuration file. It will also save model checkpoints after every 1000 steps, enabling training to be stopped and continued from the last saved checkpoint. However, it is important to note that if the model continues to be trained past the optimal point, it may start to overfit the training

data, which means it becomes too specialized and fails to generalize well to new data. Hence, it is recommended to stop the training earlier to avoid overfitting of the model.

```
%cd /content/models/research/object_detection

##Export inference graph
!python exporter_main_v2.py --
trained_checkpoint_dir=/mydrive/customtf/training2 \
--pipeline_config_path={pipeline_file} \
--output_directory /mydrive/customtf/data/inference_graph
```

Figure 4.17 Validation during training

While training is ongoing, the "export\_main\_v2.py" Python script shown in Figure 4.17 can be used in another Google Colab notebook to validate the current status of the model using the validation datasets and record.

```
INFO:tensorflow:Eval metrics at step 22000
I0420 20:26:46.526081 139741908993856 model_lib_v2.py:1015] Eval metrics at step 22000
INFO:tensorflow:      + DetectionBoxes_Precision/mAP: 0.908195
I0420 20:26:46.534008 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP: 0.908195
INFO:tensorflow:      + DetectionBoxes_Precision/mAP@.50IOU: 0.978336
I0420 20:26:46.535499 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.50IOU: 0.978336
INFO:tensorflow:      + DetectionBoxes_Precision/mAP@.75IOU: 0.977673
I0420 20:26:46.536776 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.75IOU: 0.977673
INFO:tensorflow:      + DetectionBoxes_Precision/mAP (small): -1.000000
I0420 20:26:46.538003 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow:      + DetectionBoxes_Precision/mAP (medium): 0.400000
I0420 20:26:46.539158 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (medium): 0.400000
INFO:tensorflow:      + DetectionBoxes_Precision/mAP (large): 0.912156
I0420 20:26:46.540360 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (large): 0.912156
INFO:tensorflow:      + DetectionBoxes_Recall/AR@1: 0.931468
I0420 20:26:46.541609 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@1: 0.931468
INFO:tensorflow:      + DetectionBoxes_Recall/AR@10: 0.932950
I0420 20:26:46.542878 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@10: 0.932950
INFO:tensorflow:      + DetectionBoxes_Recall/AR@100: 0.932950
I0420 20:26:46.543919 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100: 0.932950
INFO:tensorflow:      + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0420 20:26:46.544954 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow:      + DetectionBoxes_Recall/AR@100 (medium): 0.500000
I0420 20:26:46.546122 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (medium): 0.500000
INFO:tensorflow:      + DetectionBoxes_Recall/AR@100 (large): 0.934802
I0420 20:26:46.547563 139741908993856 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (large): 0.934802
INFO:tensorflow:      + Loss/localization_loss: 0.013974
I0420 20:26:46.548546 139741908993856 model_lib_v2.py:1018] + Loss/localization_loss: 0.013974
INFO:tensorflow:      + Loss/classification_loss: 0.072544
I0420 20:26:46.549531 139741908993856 model_lib_v2.py:1018] + Loss/classification_loss: 0.072544
INFO:tensorflow:      + Loss/regularization_loss: 0.060509
I0420 20:26:46.550524 139741908993856 model_lib_v2.py:1018] + Loss/regularization_loss: 0.060509
INFO:tensorflow:      + Loss/total_loss: 0.147027
I0420 20:26:46.551521 139741908993856 model_lib_v2.py:1018] + Loss/total_loss: 0.147027
```

Figure 4.18 Evaluation result

The evaluation of the model using evaluation data at step 22,000 is presented in Figure 4.18, which displays the average precision and recall for different Intersection over Union sizes. The average precision refers to the percentage of correctly detected objects among all detected objects, while the average recall refers to the percentage of correctly detected objects among all actual objects. Therefore, a high average precision means that the model has a low number of false positives, while a high average recall means that the

model has a low number of false negatives. The model achieved an average precision of 0.91 and an average recall of 0.93, indicating that it has a high accuracy in detecting objects in images. Figure 4.18 highlights the low total loss of the model during its evaluation, which was 0.14.

```
INFO:tensorflow:Step 22000 per-step time 0.514s
I0420 20:22:36.032456 140404840134464 model_lib_v2.py:705] Step 22000 per-step time 0.514s
INFO:tensorflow:{'Loss/classification_loss': 0.029743537,
'Loss/localization_loss': 0.0067845485,
'Loss/regularization_loss': 0.060510993,
'Loss/total_loss': 0.09703907,
'learning_rate': 0.048900835}
```

Figure 4.19 Stop model training

At step 22,000 of the training, the model loss and its learning rate are depicted in Figure 4.19. The figure illustrates that the total loss of the model during training has decreased to 0.09, which is considered relatively low as it is below 0.1. Furthermore, the decrease in learning rate to 0.04 suggests that the model is now learning at a slower pace. This could lead to overfitting if the training is continued further. As a result, the training process was stopped at step 22,000 to prevent overfitting of the model.

```
# Make a directory to store the trained TFLite model
# !mkdir /mydrive/customtf/custom_model_lite/
!mkdir /mydrive/customtf/custom_model_lite/ssd
output_directory = '/mydrive/customtf/custom_model_lite/ssd'

# Path to training directory (the conversion script automatically chooses the highest ch
eckpoint file)
last_model_path = '/mydrive/customtf/training3'

!python /content/models/research/object_detection/export_tflite_graph_tf2.py \
  --trained_checkpoint_dir {last_model_path} \
  --output_directory {output_directory} \
  --pipeline_config_path {pipeline_file}
```

Figure 4.20 Export Inference Graph

Once the training has been stopped, the next step is to convert the model into an inference graph, which is a representation of the trained model that can be used for detection or inference. This can be done by exporting the inference graph and storing it in a new directory. In Figure 4.20, this is achieved using the "!mkdir" command to create the directory, and the "export\_tflite\_graph\_tf2.py" script to export the inference graph to that directory. The inference graph is important because it allows the model to be used in a

production environment, such as a mobile application, where it can make detections on new data.

```
# Convert exported graph file into TFLite model file
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model('/mydrive/customtf/custom_model_lite/ssd/saved_model')
tflite_model = converter.convert()

with open('/mydrive/customtf/custom_model_lite/ssd/detect_ssd.tflite', 'wb') as f:
    f.write(tflite_model)
```

Figure 4.21 Create tflite file

The highlighted command in Figure 4.21 demonstrates the process of creating a TFLite file from the imported inference graph. The TensorFlow function called "TFLiteConverter" is utilized to perform this conversion, resulting in a new TFLite file named "detect\_ssd.tflite".

```
# Set up variables for running user's model
PATH_TO_IMAGES= /mydrive/customtf/test' # Path to test images folder
PATH_TO_MODEL= '/mydrive/customtf/custom_model_lite/ssd/detect_ssd.tflite'
# Path to .tflite model file
PATH_TO_LABELS= '/mydrive/customtf/labelmap.txt' # Path to labelmap.txt file
min_conf_threshold=0.5
# Confidence threshold(try changing this to 0.01 if you don't see any detection results)
images_to_test = 10 # Number of images to run detection on

# Run inferencing function!
tflite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold,
images_to_test)
```

Figure 4.22 Model testing command

The command displayed in Figure 4.22 demonstrates how to test the model "detect\_ssd.tflite" using images from the test folder. To perform the detection, the "tflite\_detect\_images" command provided by TensorFlow will be used to load the model and the images.

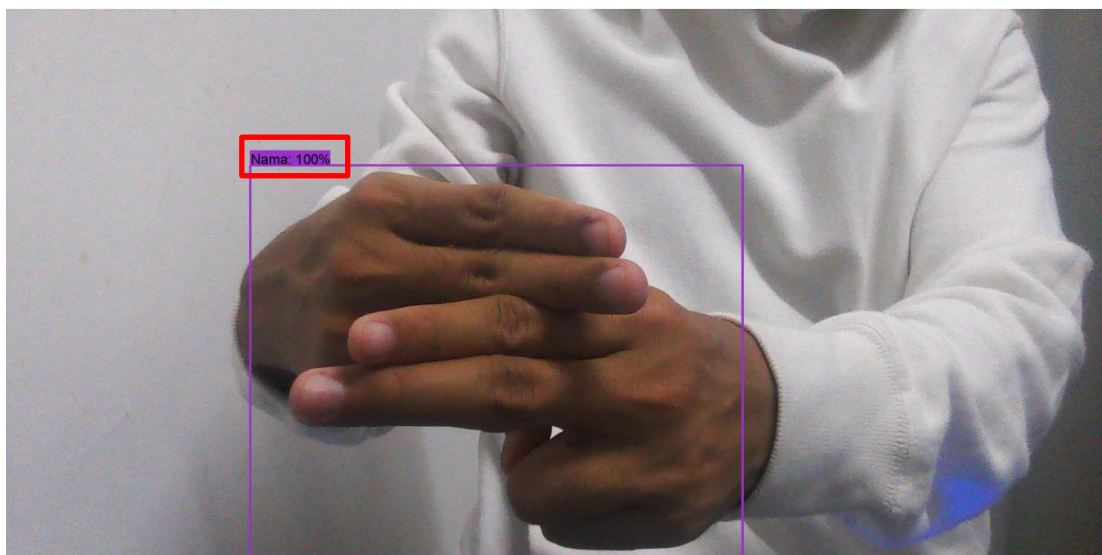


Figure 4.23 Testing model

The output of the SSD-MobileNet model detection is displayed in Figure 4.23. The model was able to accurately detect the class label, which in this case is "Nama," with a confidence score of 100%. This detection process was repeated for all the classes that were trained in the model.

```
files.download('/mydrive/customtf/custom_model_lite.zip')
```

Figure 4.24 Download TFLite file

The TFLite file can be downloaded by executing the command shown in Figure 4.24 once the training and testing results are deemed satisfactory.



Figure 4.25 Downloaded TFLite model

Figure 4.25 displayed the “detect\_ssd.tflite” file that has been downloaded. This file will be used for real-time testing in android mobile application.

## 4.2.4 Real-time testing

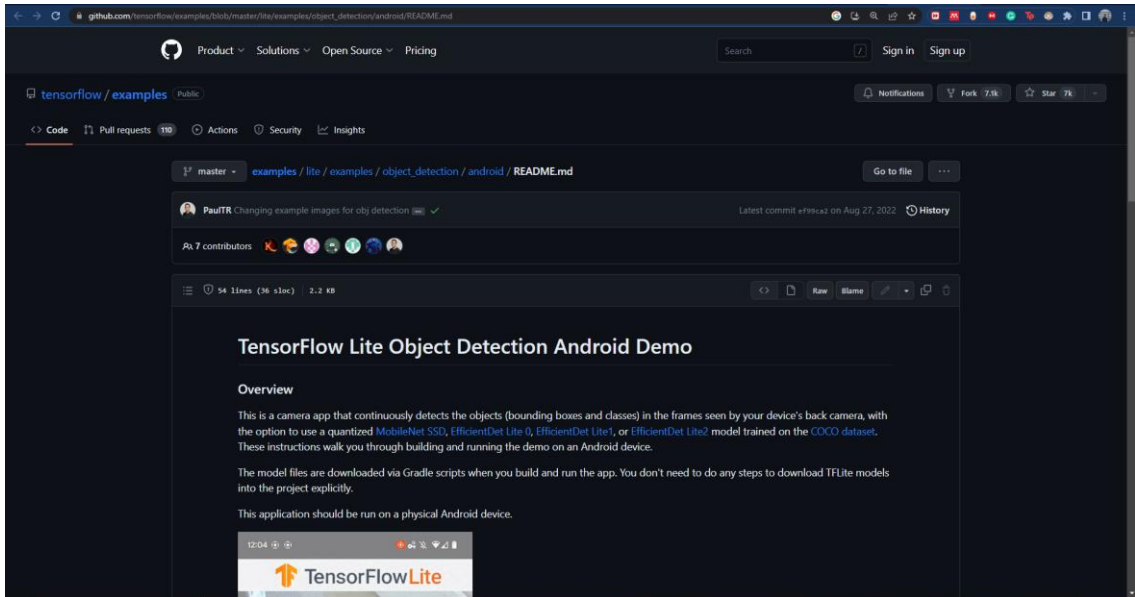


Figure 4.26 Tensorflow Mobile apps Github

The GitHub page for the Tensorflow Lite object detection Android mobile application is displayed in Figure 4.26. Cloning the repository allows for the replacement of the pre-existing model in the Tensorflow mobile application with the newly trained SSD model.

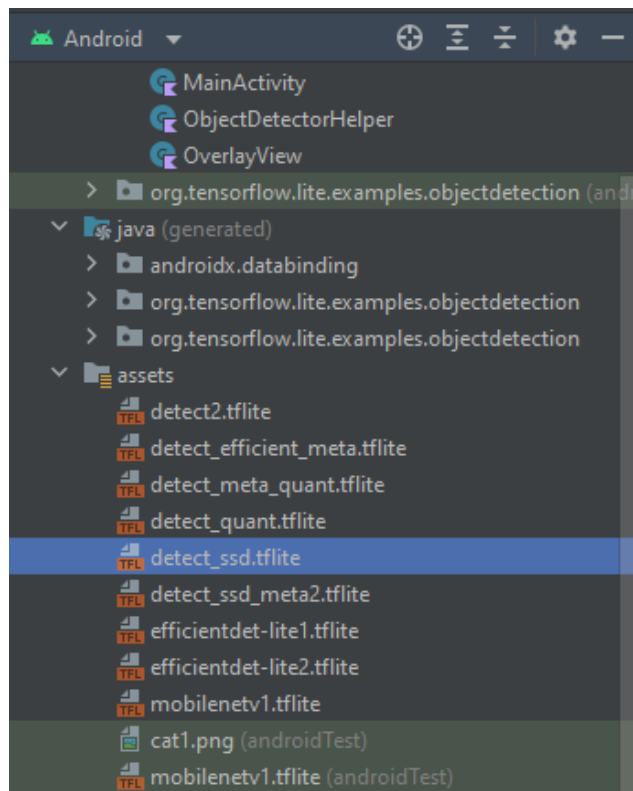
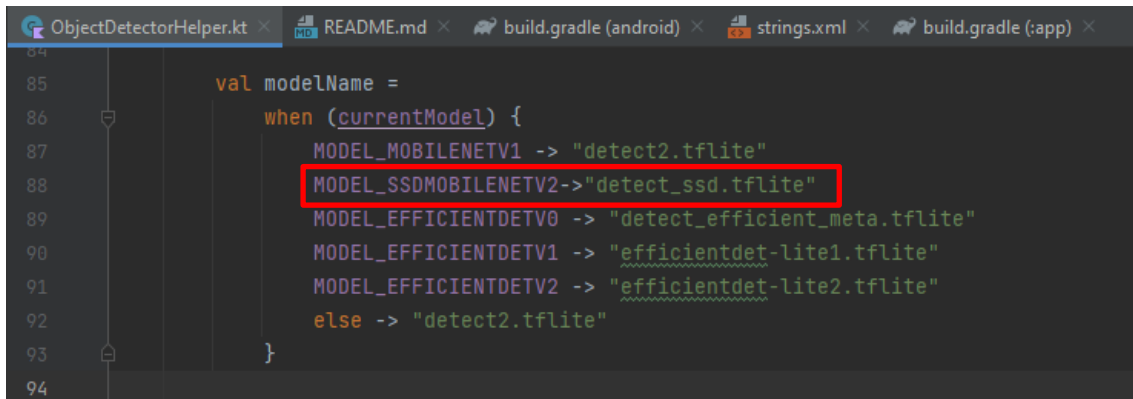


Figure 4.27 Copy model into android asset

After opening the cloned Tensorflow mobile application project in Android Studio. The "detect\_ssd.tflite" model was copied into the assets folder of the Android studio project as depicted in figure 4.27. By copying the model into the assets folder, the application can access the model to perform real-time object detection using the trained SSD-MobileNet model. This process is a crucial step towards deploying the trained model for detecting objects in real-time images captured using a mobile device.



```
84  
85     val modelName =  
86         when (currentModel) {  
87             MODEL_MOBILENETV1 -> "detect2.tflite"  
88             MODEL_SSDMOBILENETV2 -> "detect_ssd.tflite"  
89             MODEL_EFFICIENTDET0 -> "detect_efficient_meta.tflite"  
90             MODEL_EFFICIENTDET1 -> "efficientdet-lite1.tflite"  
91             MODEL_EFFICIENTDET2 -> "efficientdet-lite2.tflite"  
92             else -> "detect2.tflite"  
93         }  
94
```

Figure 4.28 Change model name

Figure 4.28 illustrates how the name of the tflite model is set as "detect\_ssd.tflite" in the "ObjectDetectorHelper.kt" file. By modifying the model name, the application can utilize the trained SSD model for real-time detection of Malay Sign language.

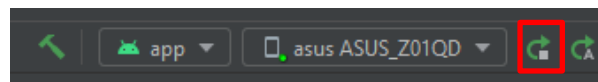


Figure 4.29 Install mobile application

Upon clicking the "run" button as shown in Figure 4.29, the mobile application will be installed onto the selected mobile phone.

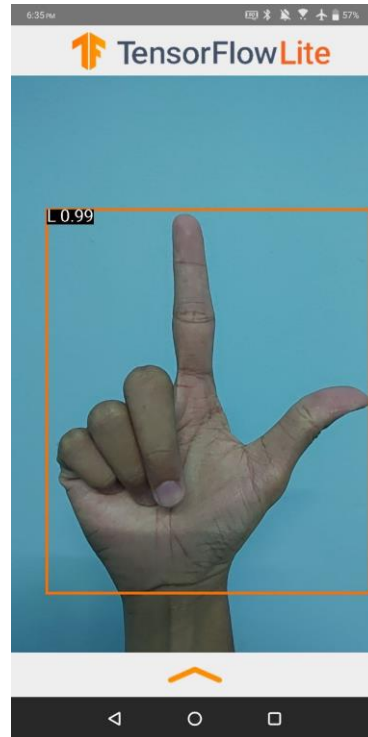


Figure 4.30 Real-time Testing in mobile apps

The real-time testing of the installed mobile application on an Android phone is depicted in Figure 4.30. The figure shows that the trained model was able to accurately detect the "L" sign with a high confidence score of 0.99 and drew a bounding box perfectly around it. The testing results will be recorded in the test plan document that have been prepared.



## 4.3 Result

The developed object detection model of SSD MobileNet was tested and compared with other models that have been developed using the same method and same number of training steps which are 20,000. The result of the SSD-MobileNet testing is evaluated and recorded for analysis.

### 4.3.1 Model evaluations



Figure 4.31 Total loss graph SSD-MobileNet model

Figure 4.31 displayed the graph of total loss versus number of steps for SSD-MobileNet model during training. Based on the figure, the model able to achieve a relatively low total loss of below 0.14 at the 20,000 steps when the model stopped training.

```
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.908
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.978
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.978
```

Figure 4.32 Average Precision result after training

```
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.931
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.933
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.933
```

Figure 4.33 Average Recall result after training

```
INFO:tensorflow: + Loss/total_loss: 0.147027
I0514 13:24:38.962524 140193166391104 model_lib_v2.py:1018] + Loss/total_loss: 0.147027
```

Figure 4.34 Total loss result after training

Once the training process is completed, the model can be evaluated using the prepared evaluation data. The evaluation is carried out using the Python script provided by TensorFlow. Figures 4.32, 4.33, and 4.34 present the evaluation results obtained from the evaluation data.

In Figure 4.32, the model achieved an average precision score of 0.908, indicating its ability to accurately detect and classify objects. Figure 4.33 demonstrates that the model achieved an average precision of 0.931, further confirming its effectiveness in object detection. Additionally, Figure 4.34 highlights the low total loss value of 0.147027, indicating that the model has effectively learned from the training data.

Table 4.2 Model evaluation result

Model	SSD-MobileNet-v2	SSD-Efficientdet-d0	SSD-Resnet50
Average Precision	<b>0.908</b>	0.411	0.906
Average Recall	<b>0.931</b>	0.680	0.904
Total loss	<b>0.147027</b>	0.911097	0.203545

Table 4.2 shows the results of evaluation for all the object detection model that have been developed. Average precision is a measure of the quality of the detection algorithm, based on the precision and recall of the algorithm over a range of threshold values. Precision is the fraction of true positives out of all detected positives, while recall is the fraction of true positives out of all actual positives. A high average precision indicates that the model is able to correctly identify objects with minimal false positives.

Out of the three models, SSD-MobileNet-v2 achieved the highest average precision of 0.908. This indicates that the model is able to detect objects accurately with minimal false positives, resulting in high precision and recall. This is because SSD-MobileNet-v2 uses a lightweight MobileNet-v2 architecture for feature extraction, making it more efficient and faster than other models. SSD-Efficientdet-d0 had the lowest average precision of 0.411, indicating that it performed poorly compared to the other models. This could be due to the fact that it uses a complex architecture that requires more computational resources and longer training time. SSD-Resnet50 achieved an average precision of 0.906, which is slightly lower than SSD-MobileNet-v2 but still a good performance. However, SSD-Resnet50 requires more computational resources and longer training time compared to SSD-MobileNet-v2.

When it comes to object detection models, average recall is an important metric to consider because it indicates how well the model is able to correctly detect objects in the input data. Recall measures the proportion of actual positive instances that are correctly identified by the model. In other words, it represents the model's ability to identify all

relevant objects in the image. In this comparison, we can see that the SSD-MobileNet-v2 has the highest average recall score of 0.931, followed by SSD-Resnet50 at 0.904, and SSD-Efficientdet-d0 with 0.680. This means that the SSD-MobileNet-v2 model is able to detect a higher proportion of the relevant objects in the input data compared to the other models.

Total loss is a commonly used evaluation metric in object detection models that indicates the overall loss of the model during training. The lower the total loss, the better the model performs during training. A high total loss means the model is struggling to learn and is not able to accurately detect objects in the images. In terms of the listed models, SSD-MobileNet-v2 has the lowest total loss of 0.147027, which indicates that it performed better during training than the other models. On the other hand, SSD-Efficientdet-d0 has a significantly higher total loss of 0.911097, indicating that the model had more difficulty learning during the training process. SSD-Resnet50 has a total loss of 0.203545, which is higher than SSD-MobileNet-v2 but lower than SSD-Efficientdet-d0. This suggests that the model was able to learn and perform reasonably well during training, but not as well as the SSD-MobileNet-v2.

Overall, the evaluation and comparison of the three object detection models - SSD-MobileNet-v2, SSD-Efficientdet-d0, and SSD-Resnet50 - show that SSD-MobileNet-v2 performs the best in terms of both precision and recall, while also having the lowest total loss. This suggests that SSD-MobileNet-v2 is the most accurate and efficient model for detecting sign language gestures in real-time images captured using a mobile device. However, it is important to note that further research and optimization may be needed to improve the performance of the models in different environments and with different types of sign language gestures.

### 4.3.2 Model Testing

Once the model was converted into TFLite type, it underwent testing using the Mean Average Precision (mAP) formula on the prepared test data. Converting the model to TFLite type result in a slight decrease in its performance, as TFLite is designed to be lightweight and requires less computational power with higher latency.

```
***mAP Results***
```

Class	Average mAP @ 0.5:0.95
A	88.00%
B	92.50%
C	86.88%
D	100.00%
E	82.08%
F	90.50%
G	93.33%
H	87.50%
I	92.22%
K	83.33%
L	91.11%
M	91.11%
N	91.60%
O	93.33%
P	92.50%
Q	67.25%
R	94.87%
S	90.67%
T	97.50%
U	74.00%
V	90.36%
W	94.29%
X	96.40%
Y	98.57%
Saya	94.44%
Umur	87.78%
Nama	93.67%
1	96.67%
2	92.67%
3	93.33%
4	93.33%
5	100.00%
6	98.27%
7	100.00%
8	95.62%
9	92.86%
Overall	91.63%

Figure 4.35 Model Evaluation result after TFLite conversion

The evaluation results after converting the model to the TFLite format are presented in Figure 4.35. This figure displays the mAP scores for each class at various Intersection over Union (IoU) thresholds. Notably, the highlighted section indicates that the model achieved a significantly high mAP score of 91.63%. The achieved mAP score of 91.63% emphasizes the model's effectiveness in accurately detecting and classifying objects in the evaluation dataset.

Table 4.3 Model testing result

Model	SSD-MobileNet-v2	SSD-Efficientdet-d0	SSD-Resnet50
Mean Average Precision (mAP)	<b>91.63%</b>	62.45%	85.34%





Table 4.3 displays the mAP score for each model during the testing phase. mAP is the metric used to evaluate object detection models. It is the average of the precision values at different levels of recall. In other words, mAP measures how well the model can detect and localize objects of interest. The higher the mAP score, the better the model's performance.


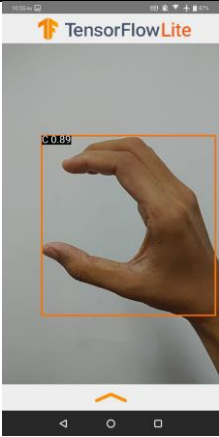

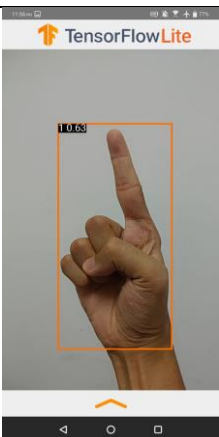


In the comparison of the three models, SSD-MobileNet-v2 has the highest mAP score of 91.63%, indicating that it is the best model in terms of object detection performance. SSD-Resnet50 comes in second with an mAP score of 85.34%, while SSD-Efficientdet-d0 has the lowest mAP score of 62.45%. This suggests that SSD-MobileNet-v2 and SSD-Resnet50 are more suitable for real-world applications that require high accuracy in object detection, while SSD-Efficientdet-d0 may be more appropriate for applications where computational resources are limited.




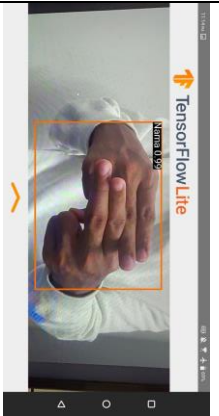

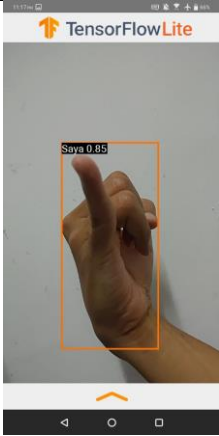
### 4.3.3 Real-Time testing

Table 4.4 presented below displays the outcomes of the real-time testing, including the detected labels and their corresponding confidence scores. It is important to note that the table solely includes the results for three letters, three numbers, and three Malay phrases. For the complete set of testing results, please refer to Appendix A.


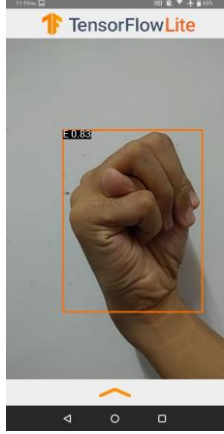
Table 4.4 Real-time testing result

No.	Sign Image	Expected Signs label	Output Image	Output label	Output Confidence score
1		A		A	0.84
2		B		B	0.87

3	 <p>C</p>	C		C	0.89
4	 <p>1</p>	1		1	0.63
5	 <p>2</p>	2		V	0.99

6	 <p>3</p>	3		W	0.81
7		Nama		Nama	0.99
8		Saya		Saya	0.85



9		Umur		E	0.83
---	---	------	--	---	------

Based on the outcomes of the real-time testing, a corresponding confusion matrix (Table 4.5) has been generated.

Table 4.5 Real-time Testing result

	Actually Positive	Actually Negative
<i>Detected Positive</i>	TP=29	FP=0
<i>Detected Negative</i>	FN=1	TN=7

TP= true positive, FP=false positive, FN=false negative, TN= true negative

Table 4.5 shows the confusion matrix from the real-time testing, this result was used to calculate the accuracy, recall, precision, and F1-score of the developed SSD-MobileNet model.

Table 4.6 Evaluation matrix result

Model	SSD-MobileNet-v2	SSD-Efficientdet-d0	SSD-Resnet50
Accuracy	<b>0.97</b>	0.68	0.86
Precision	<b>1</b>	0.63	0.83
Recall	<b>0.96</b>	0.67	0.85
F1-Score	<b>0.97</b>	0.64	0.83

Table 4.6 presents the computed accuracy, precision, recall, and F1-score obtained from real-time testing on Android mobile applications. In terms of accuracy, SSD-MobileNet-v2 achieved the highest accuracy score of 0.97, indicating its ability to correctly classify objects in the evaluation dataset. SSD-Resnet50 followed closely behind with an accuracy of 0.86, while SSD-Efficientdet-d0 lagged behind with an accuracy of 0.68. This suggests that SSD-MobileNet-v2 outperforms the other models in accurately identifying objects.

Precision measures the model's ability to correctly identify positive instances among all the instances it detects as positive. Here, SSD-MobileNet-v2 achieved a perfect precision score of 1, indicating that all its positive detections were indeed correct. SSD-Resnet50 achieved a precision score of 0.83, while SSD-Efficientdet-d0 had the lowest precision score of 0.63. This suggests that SSD-MobileNet-v2 excels in making precise detections.

Recall measures the model's ability to identify all positive instances among all the actual positive instances. SSD-MobileNet-v2 achieved a recall score of 0.96, indicating its capability to capture a high proportion of positive instances. SSD-Resnet50 followed closely with a recall score of 0.85, while SSD-Efficientdet-d0 had a recall score of 0.67. This indicates that SSD-MobileNet-v2 has a higher sensitivity in identifying positive instances.

The F1-score combines both precision and recall into a single metric, providing an overall assessment of the model's performance. SSD-MobileNet-v2 achieved the highest F1-score of 0.97, indicating a harmonious balance between precision and recall. SSD-Resnet50 had an F1-score of 0.83, while SSD-Efficientdet-d0 had the lowest F1-score of 0.64. This suggests that SSD-MobileNet-v2 is the most balanced model in terms of precision and recall.

Overall, based on the comparison and analysis of accuracy, precision, recall, and F1-score, SSD-MobileNet-v2 emerges as the superior model in object detection. It achieves high accuracy, perfect precision, high recall, and a balanced F1-score. These results highlight the model's robustness and effectiveness in accurately detecting and classifying objects. Therefore, SSD-MobileNet-v2 is recommended for applications that require high precision and recall in object detection tasks.

#### **4.4 Discussion**

The SSD-MobileNet model has shown promising results in sign language detection, achieving a high average precision, recall, and mAP score. The model has been implemented in real-time detection of Malay sign language by converting the model to TFLite type for deployment on mobile devices with limited computational power. However, to make the model more accurate and effective in detecting Malay sign language, the model could be trained with a larger dataset of Malay sign language gestures. This will ensure that the model can detect and recognize a wide range of gestures in Malay sign language, leading to higher accuracy and precision. In conclusion, SSD-MobileNet-v2 is the suitable model for sign language detection based on its high average precision and recall scores, low total loss, accuracy, F1-score, and high mAP score. The model's performance is critical for accurate detection of sign language, which is essential for effective communication for the hearing-impaired community.

## **CHAPTER 5**

### **CONCLUSION**

#### **5.1 Introduction**

In this research, a sign language object detection model was developed using transfer learning. The SSD-MobileNet-v2 model was trained on sign language datasets to enable real-time detection. The training process involved utilizing Python and Android Studio to achieve a high accuracy score. The performance of the SSD-MobileNet-v2 model was compared with two other models, namely SSD-Resnet50 and SSD-Efficientdet-D0. The development process for these models followed a similar methodology as that of the SSD-MobileNet-v2 model. The evaluation and testing results were analyzed and compared to gain a deeper understanding. The findings indicate that the SSD-MobileNet-v2 method is the most effective for real-time detection and classification of sign languages, as it achieves the highest accuracy and precision.

#### **5.2 Objective Revisited**

As discussed in chapter 1, there are three objectives that need to be achieved at the end of this research. The first objective is to study the usage of object detection using Single Shot Detect (SSD) and MobileNet in studying sign language. This objective was achieved by developing an SSD-MobileNet model capable of real-time sign language detection, as discussed in detail in Chapter 4.

The second objective involved designing and training an object detection model using Single Shot Detector and MobileNet architectures to study sign language. This objective was accomplished by setting up the environment in Google Colab using Python and utilizing a pre-trained SSD-MobileNet model provided by TensorFlow. The process of designing and training the model was elaborated upon in Chapter 4.

The final objective focused on evaluating the functionality of the developed object detection model in studying and detecting sign language. This evaluation was conducted using evaluation metrics discussed in Chapter 3, which included multiple tests to assess the performance of the developed SSD-MobileNet model by analyzing its accuracy, precision, recall, and F1-score compared to other models. Further details regarding the evaluation results were extensively discussed in Chapter 4.

In conclusion, this research successfully achieved all stated objectives. The implementation of an object detection model for real-time sign language detection demonstrated the superiority of the SSD-MobileNet model, as it exhibited the highest precision, accuracy, F1-score, and recall scores when compared to other models.

### **5.3 Limitation**

The testing results presented in Chapter 4 demonstrated promising outcomes, surpassing a satisfactory accuracy threshold of 90%. Furthermore, the comparison with other models highlighted the potential of real-time sign language detection. However, a notable limitation in this research is the scarcity of available datasets for various signs beyond numbers and letters. Consequently, the model's training was constrained to only three phrases, namely "Nama," "Saya," and "Umur." These signs were manually collected, resulting in reduced data diversity and volume.

Another constraint of the model is its inability to detect dynamic sign languages, which involve hand movements in combination. Presently, the model is only capable of detecting static signs, thereby limiting its ability to learn and recognize a broader range of sign languages. One additional limitation is the latency exhibited by the models, despite the developed model being considered lightweight. It is important to note that TensorFlow is continuously working on enhancing and optimizing the capabilities of TensorFlow Lite models to improve efficiency and reduce computational requirements. During real-time testing on an Android phone, the observed latency typically fluctuates between 70 ms to 80 ms. Moreover, it is worth mentioning that prolonged usage of the application on the phone may lead to increased heat generation.

## **5.4 Future Work**

For future work, it is crucial to prioritize the inclusion of a more diverse dataset in the study, encompassing a wide range of signs with variations in hand size, shape, and color. This approach ensures that the model learns and detects signs accurately across different variations. However, it should be noted that the data collection process is laborious and time-consuming, requiring significant effort to manually curate a large and diverse dataset of high quality.

Furthermore, future models should be designed to address the detection of dynamic sign language, which is commonly used in everyday communication. Consideration should also be given to factors such as hand positioning and facial expressions, as they play vital roles in sign language. Developing a new model that can facilitate real-time communication between hearing individuals and the deaf community would require incorporating these aspects into the model's training process.

## REFERENCES


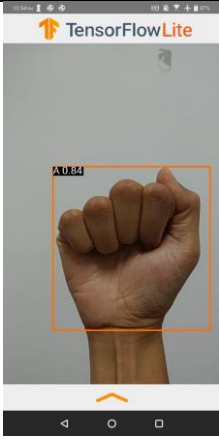



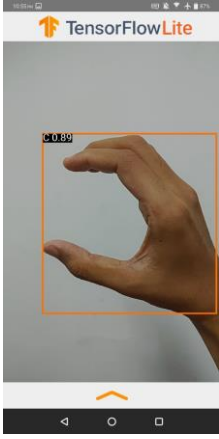
- Ammar Alhaj Ali. (2020). *American Sign Language Letters* | Kaggle. <https://www.kaggle.com/datasets/ammarnassanalhajali/american-sign-language-letters>
- Ankit Sachan. (2017). *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, YOLO, SSD – CV-Tricks.com*. <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>
- Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. <http://arxiv.org/abs/2004.10934>
- Capilla, D. M. (2012). *Sign Language Translator using Microsoft Kinect XBOX 360 TM*.
- Darus, N. M., Abdullah, N. T., & Mutalib, A. A. (2012). *iMSL: Malay Sign Language for the Deaf and Hearing-impaired*. 4–6.
- Das, S., Imtiaz, M. S., Neom, N. H., Siddique, N., & Wang, H. (2023). A hybrid approach for Bangla sign language recognition using deep transfer learning model with random forest classifier. *Expert Systems with Applications*, 213, 118914. <https://doi.org/10.1016/J.ESWA.2022.118914>
- DAYAS, I. A. (2019). *The history of sign language*. <https://www.nationalgeographic.com/history/history-magazine/article/creation-of-sign-language>
- Dianne Castillo. (2021). *Transfer Learning for Machine Learning - Seldon*. <https://www.seldon.io/transfer-learning>
- Estrada, J., Paheding, S., Yang, X., & Niyaz, Q. (2022). Deep-Learning-Incorporated Augmented Reality Application for Engineering Lab Training. *Applied Sciences (Switzerland)*, 12(10). <https://doi.org/10.3390/APP12105159>
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)*. <http://www.cs.berkeley.edu/~rbg/rcnn>.
- Grace Karimi. (2021). *Introduction to YOLO Algorithm for Object Detection | Engineering Education (EngEd) Program | Section*. <https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/>




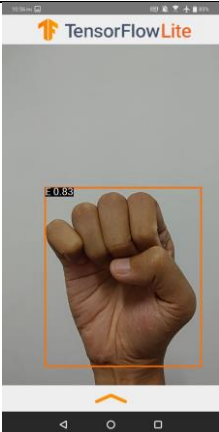


- Hmrishav Bandyopadhyay. (2022). *YOLO: Real-Time Object Detection Explained*.  
<https://www.v7labs.com/blog/yolo-object-detection>
- Jamie Berke. (2020, April 20). *Challenges of Learning Sign Language*.  
<https://www.verywellhealth.com/challenges-of-learning-sign-language-1049296>
- Jason Brownlee. (2021). *A Gentle Introduction to Object Recognition With Deep Learning*.  
<https://machinelearningmastery.com/object-recognition-with-deep-learning/>
- João Cartucho. (2018). *Cartucho/OpenLabeling: Label images and video for Computer Vision applications*. João Cartucho. <https://github.com/Cartucho/OpenLabeling>
- Krlevska, A., Trajanov, R., & Gievska, S. (2022). Real-time Macedonian Sign Language Recognition System by using Transfer Learning. *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology, MIPRO 2022 - Proceedings*, 906–911. <https://doi.org/10.23919/MIPRO55190.2022.9803692>
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). *SSD: Single Shot MultiBox Detector*. [https://doi.org/10.1007/978-3-319-46448-0\\_2](https://doi.org/10.1007/978-3-319-46448-0_2)
- Mavi, A., & Dikle, Z. (2022). *A New 27 Class Sign Language Dataset Collected from 173 Individuals*. /[datasets/ardamavi/27-class-sign-language-dataset](https://github.com/ardamavi/27-class-sign-language-dataset)
- Mittal, P., Singh, R., & Sharma, A. (2020). Deep learning-based object detection in low-altitude UAV datasets: A survey. In *Image and Vision Computing* (Vol. 104). Elsevier Ltd.  
<https://doi.org/10.1016/j.imavis.2020.104046>
- PapersWithCode. (2022). *PASCAL VOC 2007 Benchmark (Object Detection) | Papers With Code*. <https://paperswithcode.com/sota/object-detection-on-pascal-voc-2007>
- Pathak, A. R., Pandey, M., & Rautaray, S. (2018). Application of Deep Learning for Object Detection. *Procedia Computer Science*, 132, 1706–1717.  
<https://doi.org/10.1016/J.PROCS.2018.05.144>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. <http://arxiv.org/abs/1506.02640>
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. <http://image-net.org/challenges/LSVRC/2015/results>


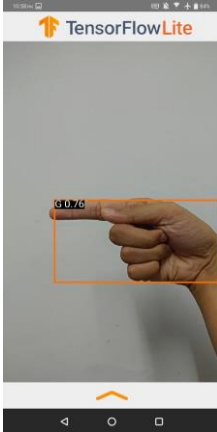

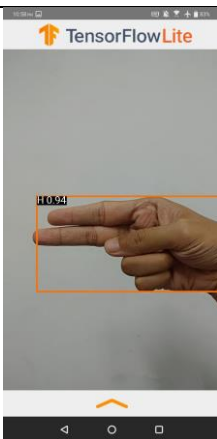

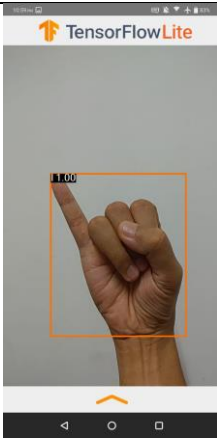







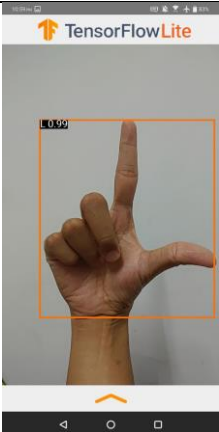


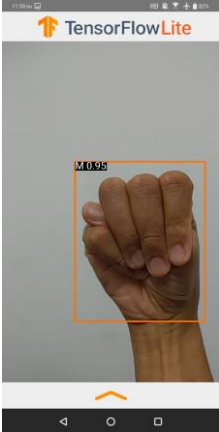
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510–4520. <https://doi.org/10.1109/CVPR.2018.00474>
- Shafi, I., Mazahir, A., Fatima, A., & Ashraf, I. (2022). Internal defects detection and classification in hollow cylindrical surfaces using single shot detection and MobileNet. *Measurement*, 202, 111836. <https://doi.org/10.1016/j.measurement.2022.111836>
- Sharif Elfouly. (2019). *R-CNN (Object Detection). A beginners guide to one of the most...* | by Sharif Elfouly | *shafu.eth* | Medium. <https://medium.com/@selfouly/r-cnn-3a9beddfd55a>
- Sharrab, Y. O., Alsmirat, M., Dwekat, Z., Alsmadi, I., & Al-Khasawneh, A. (2021). Performance Comparison of Several Deep Learning-Based Object Detection Algorithms Utilizing Thermal Images; Performance Comparison of Several Deep Learning-Based Object Detection Algorithms Utilizing Thermal Images. *2021 Second International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. <https://doi.org/10.1109/IDSTA53674.2021.9660820>
- Siti Hajar, J. (2019, May 3). *Dua juta rakyat Malaysia ada masalah pendengaran - Kajian / Astro Awani*. <https://www.astroawani.com/gaya-hidup/dua-juta-rakyat-malaysia-ada-masalah-pendengaran-kajian-206706>
- Subbiah, U., Parameswaran, L., Kavin Kumar, D., & Kumar Thangavel, S. (2020). *An Extensive Study and Comparison of the Various Approaches to Object Detection using Deep Learning; An Extensive Study and Comparison of the Various Approaches to Object Detection using Deep Learning*.
- Tan, L., Huangfu, T., & Wu, L. (2021). *Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification*. <https://doi.org/10.21203/rs.3.rs-668895/v1>
- Tensorflow. (2015). *TensorFlow*. [https://www.tensorflow.org/?gclid=Cj0KCQjwk5ibBhDqARIsACzmgLRe4HS39LgxE2JTTHzfJAscSDuLW\\_5L4kr8P0qX5G02OvEcPZ\\_VAfIaAvd4EALw\\_wcB](https://www.tensorflow.org/?gclid=Cj0KCQjwk5ibBhDqARIsACzmgLRe4HS39LgxE2JTTHzfJAscSDuLW_5L4kr8P0qX5G02OvEcPZ_VAfIaAvd4EALw_wcB)
- Zeiler, M. D., & Fergus, R. (2014). *LNCS 8689 - Visualizing and Understanding Convolutional Networks*.


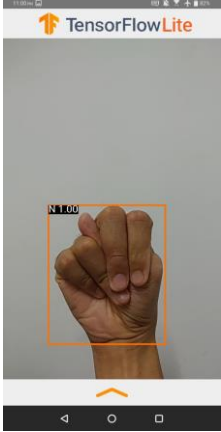

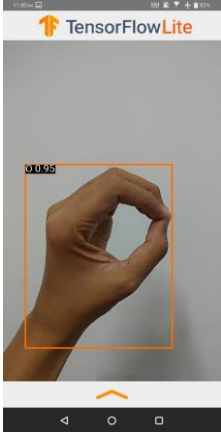

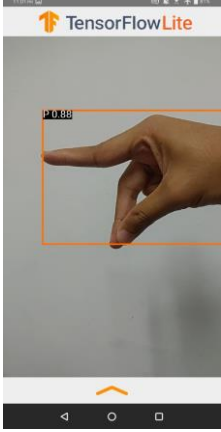
## APPENDIX A


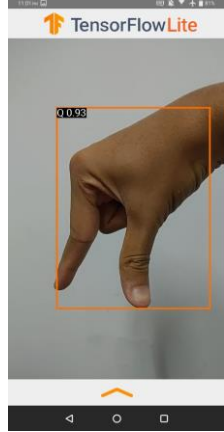

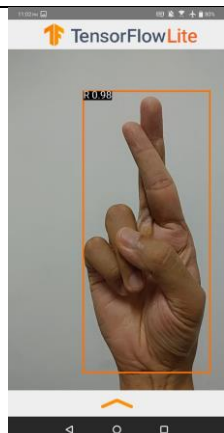


No.	Sign Image	Expected Signs label	Output Image	Output label	Output Confidence score
1	 <p style="text-align: center;">a</p>	A		A	0.84
2	 <p style="text-align: center;">b</p>	B		B	0.87
3	 <p style="text-align: center;">c</p>	C		C	0.89




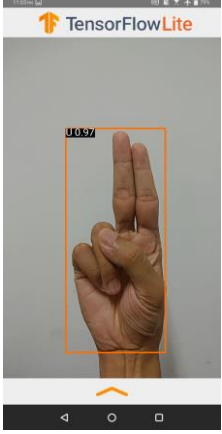


4	 <p>d</p>	D		D	0.94
5	 <p>e</p>	E		E	0.83
6	 <p>f</p>	F		F	1.00

7	 <p data-bbox="453 349 485 394">g</p>	G		G	0.76
8	 <p data-bbox="459 857 491 902">h</p>	H		H	0.94
9	 <p data-bbox="456 1375 472 1420">i</p>	I		I	1.00


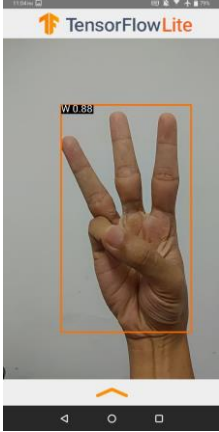



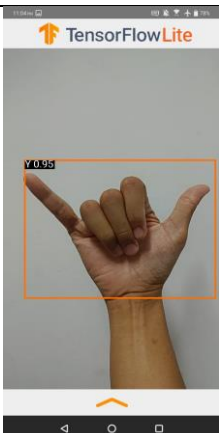
10	 	K		K	0.95
11	 	L		L	0.99
12	 	M		M	0.95


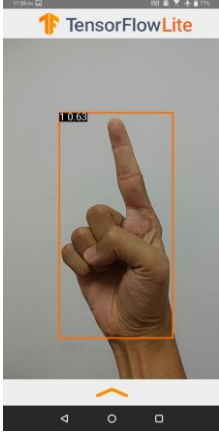




13	 <p>n</p>	N		N	1.00
14	 <p>o</p>	O		O	0.95
15	 <p>p</p>	P		P	0.88







16	 <p>q</p>	Q		Q	0.93
17	 <p>r</p>	R		R	0.98
18	 <p>S</p>	S		S	0.98







19	 t	T		T	1.00
20	 u	U		U	0.97
21	 v	V		V	0.99


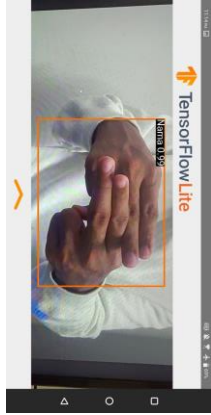

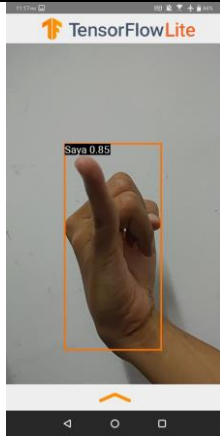

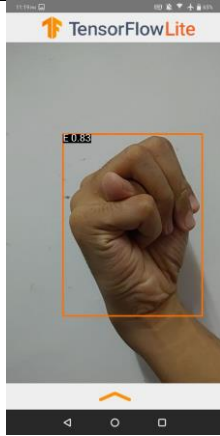


22	 <p data-bbox="443 376 486 414">W</p>	W		W	0.88
23	 <p data-bbox="459 857 486 891">X</p>	X		X	0.91
24	 <p data-bbox="459 1373 486 1406">Y</p>	Y		Y	0.95

25	 <p>1</p>	1		1	0.63
26	 <p>2</p>	2		V	0.99
27	 <p>3</p>	3		W	0.81

28	 <p>4</p>	4		4	0.53
29	 <p>5</p>	5		F	0.82
30	 <p>6</p>	6		W	0.86

1	 <p>7</p>	7		W	0.70
32	 <p>8</p>	8		8	0.96
33	 <p>9</p>	9		F	0.95

		Nama		Nama	0.99
35		Saya		Saya	0.85
36		Umur		E	0.83