

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

American Sign Language Detection

A thesis submitted in partial fulfillment of the requirements

For the degree of Master of Science in Computer Science

By

Milan Patel

May 2023

The thesis of Milan Patel is approved by:

Dr. Taehyung Wang

Date

Dr. Mahdi Ebrahimi

Date

Dr. Kyle Dewey, Chair

Date

California State University, Northridge

Acknowledgements

I would like to express my heartfelt gratitude to my advisor Dr. Kyle Dewey for the continuous support of my Master's Study and Research, for his patience, motivation, Enthusiasm and immense knowledge.

I would also like to thank my thesis committee members Dr. Taehyung Wang and Dr. Mahdi Ebrahimi for their support and guidance.

Table of Contents

Signature Page	ii
Acknowledgments	iii
List of Figures	vi
Abstract	viii
Chapter 1: Introduction.....	1
1.1 Background.....	1
1.2 Problem Statement.....	2
1.3 Motivation.....	3
Chapter 2: Dataset.....	4
2.1 Dataset Overview.....	4
2.2 Dataset Acquisition.....	5
Chapter 3: Performance Matrix.....	11
Chapter 4: Methodology.....	13
4.1 Machine Learning Models.....	13
4.2 Models Used.....	15
4.2.1 SSD MobileNet V2.....	16
4.2.1.i SSD MobileNet V2 Results.....	19
4.2.2 YOLO.....	23
4.2.2.1 Training YOLO model.....	24
4.2.3 Faster R-CNN.....	27
4.2.3.1 Faster R-CNN Results.....	29

4.3 Training Parameter.....	30
Chapter 5: Comparing the results.....	32
Chapter 6: Summary.....	35
6.1 Summary.....	35
6.2 Conclusion.....	35
6.3 Learning Experience.....	36
6.4 Future Work.....	36
References:	37

List of Figures

Figure 1: LabelImg tool used for labeling images.....	5
Figure 2: LabelImg tool used for labeling images.....	5
Figure 3: Python Script Dataset Output.....	6
Figure 4: Python Script Dataset Output.....	6
Figure 5: American Sign Language(ASL) alphabets.....	7
Figure 6: Hand Detector Module by CVZone for dataset acquisition.....	8
Figure 7: Hand Detector Module by CVZone for dataset acquisition.....	9
Figure 8: XML File Example.....	10
Figure 9: Confusion Matrix.....	12
Figure 10: SSD MobileNet V2 Architecture.....	17
Figure 11: SSD MobileNet V2 Layers.....	18
Figure 12: Label Map.....	19
Figure 13: SSD MobileNet V2 Model Training.....	20
Figure 14: SSD MobileNet V2 Result.....	20
Figure 15: SSD MobileNet V2 Model Training Iteration 2.....	21
Figure 16: SSD MobileNet V2 Result.....	22
Figure 17: YOLO Architecture.....	23
Figure 18: XML File Example.....	25
Figure 19: PascalVoc File Example.....	25
Figure 20: YOLO Result.....	26
Figure 21: YOLO Result.....	26

Figure 22: Faster R-CNN Architecture.....	27
Figure 23: Faster R-CNN Results.....	28
Figure 24: Faster R-CNN Results.....	28
Figure 25: Faster R-CNN Results.....	29
Figure 26: Performance Metrics of the Models.....	32
Figure 27: Confusion Matrix of Models.....	32
Figure 28: F-1 Confidence Curve.....	32
Figure 29: Training graphs of Loss of models.....	33
Figure 30: Training graphs of Loss of models.....	33

Abstract

American Sign Language Detection Using Different Machine Learning Models

By

Milan Patel

Master of Science in Computer Science

This research explores the application of machine learning techniques to detect American Sign Language(ASL). American Sign Language(ASL) is a complex form of communication as it involves the use of gestures, facial expressions, and body movements to communicate. The ability to detect and interpret ASL accurately and in real-time can improve communication and accessibility for individuals who are deaf/ Hard of hearing.

Chapter 1: Introduction

1.1 Background:

American sign language is a natural language used by the deaf/ Hard of hearing communities in The United States of America and most of Anglophone Canada. American Sign Language(ASL) is a complete and complex language that utilizes hand gestures, facial expressions, and body language to communicate.

The origins of ASL may be traced to the early 19th century, when Laurent Clerc a French educator immigrated to the United States and started instructing deaf children in sign language. The first American school for the deaf was established in Hartford, Connecticut, in 1864 by Thomas Hopkins Gallaudet and Clerc, and this act sparked the emergence of ASL as a separate language.[1]

ASL has developed and spread across the country over time, and the National Association of the Deaf officially recognized it as an official language in 1988. With estimates of up to 500,000 people in the US utilizing ASL as their major form of communication, ASL is currently widely utilized among the deaf community.

1.2 Problem Statement

Approximately one million people use American Sign Language (ASL) as their primary method of communication.[2] Most applications that support learning other languages are only for spoken languages. The aim of this research is to utilize different machine learning models for object detection such as YOLO V8, SSD Mobilnet V2, and Faster RCNN that can accurately recognize and interpret American Sign Language gestures in real time and use these models to teach individuals how to communicate in sign language and interpret what the other person is trying to communicate using sign language. This research will explore the challenges of training 3 different machine-learning models and comparing the results of how correctly they can detect American Sign Language (ASL).

An advantage of working on this research topic being that, ASL Letter Recognition can be used by anybody who wishes to learn sign language to communicate with:

- Deaf and Hard of hearing community
- Hearing children of deaf people
- Hearing adults who are becoming deaf, and
- People who learn it as a second language

1.3 Motivation

This situation happened to me when I was waiting in line to order a sandwich and coffee at a cafe, and the couples in front of me were communicating using ASL. Fortunately, the cashier was able to understand and respond using sign language, but what if the person on the other end didn't understand sign language? This inspired me to embark on a project that can serve as a bridge and a means of communication between those who use sign language and those who do not.

Chapter 2: Dataset

2.1 Dataset Overview

The dataset utilized for this research consists of over 6000 images, with a total of 26 labels (for each alphabet in the English language). Different tools and packages were used to make the dataset for American Sign Language. Such as LabelImg which is a Python package to draw bounding boxes around an image which then creates XML files for those images with the specific labels. OpenCV, CVzone, and Mediapipe were also used to make the dataset generation more efficient. OpenCV also known as Open Source Computer Vision Library is an open-source computer vision and machine learning software library which is used in a wide range of applications such as Robotics, surveillance, augmented reality, face recognition, and medical image analysis.

OpenCV also provides various tools and algorithms for image processing such as Feature detection. Object detection and recognition, etc.

Mediapipe is also an open-source framework developed by Google, which is also utilized for machine learning and computer vision. CVZone is a python library which was built on top of mediapipe, and the main aim of this python library is to simplify the process of using mediapipe by utilizing higher-level API and pipelines for common tasks. CVZone has a range of components to process and analyze video and audio data. Such as hand tracker, 60 FPS face detection, pose estimation, face mesh detection, etc.

The data collection for the dataset was done in 3 different ways.

2.2 Dataset Acquisition

i) Initial dataset consisted of images labeled manually using the Labellmg tool. Labellmg is a graphical image annotation tool that allows you to draw visual boxes around your objects in each image, it also automatically saves the XML files of your labeled images.

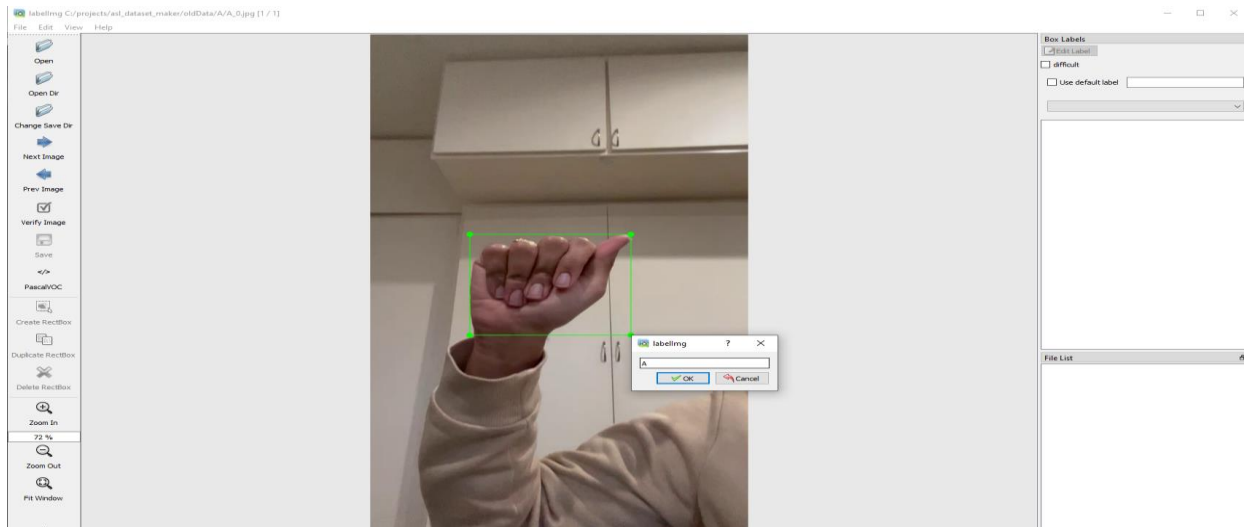


Figure 1: Labellmg tool used for labeling images

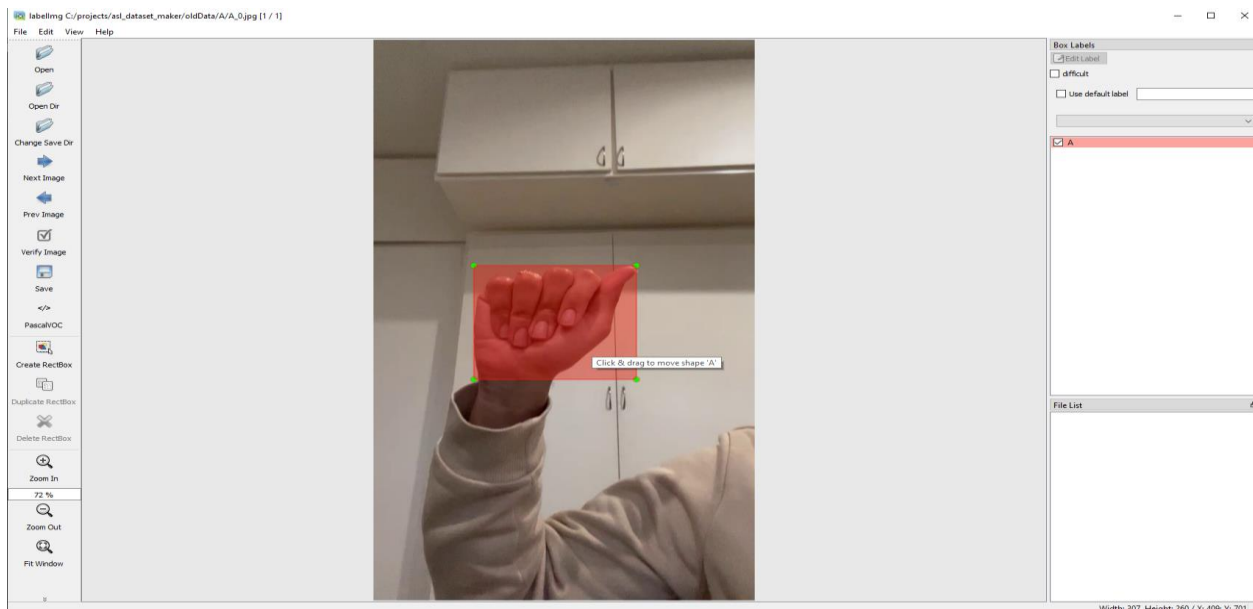


Figure 2: Labellmg tool used for labeling images

ii) Labeling images manually using the LabelImg tool started becoming difficult when I started increasing the size of the dataset to increase the precision of machine learning models. Hence, I came up with a Python script that has the bounding boxes predefined and we just have to do the sign language and the script automatically creates an XML file for that specific letter.

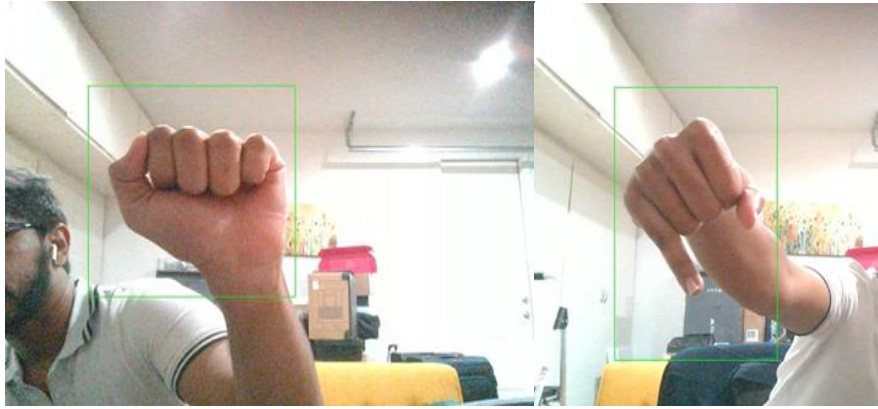


Figure 3 and 4: Python Script Dataset Output

The script has 3 different options for bounding boxes:

- 1) Square: For letters which does not require much space(letters such as A, E, G, etc)
- 2) Tall: For letters such as B, I, V, W, etc
- 3) Wide: For letters such as L, G, Y.



Figure 5: American Sign Language(ASL) alphabets [6]

iii) The previous script was further optimized, where I utilized a hand detection model which will automatically create bounding boxes and generate the XML files for respective letters. As for data collection, just upload a video of an American Sign Language gesture, and the script will automatically track the hand from the video and generate a bounding box and XML files.

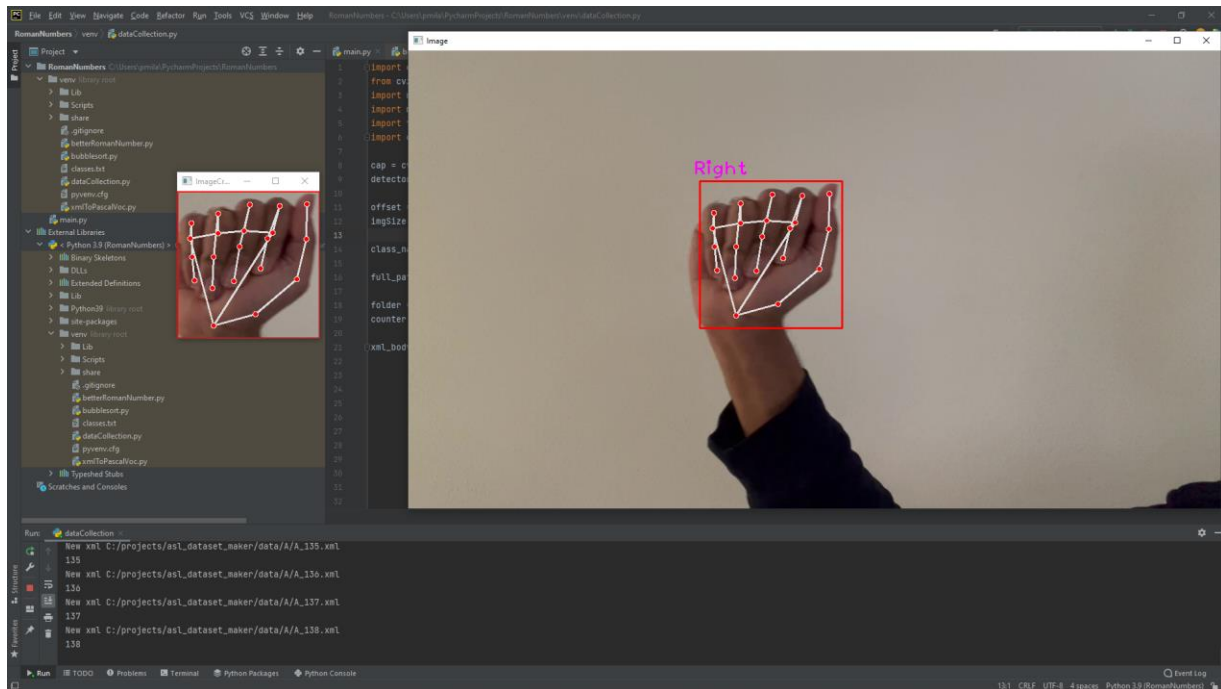


Figure 6: Hand Detector Module by CVZone for dataset acquisition

The script utilized a computer vision package that makes it easy to run image processing and AI functions. The core of the package uses OpenCV and mediapipe libraries to detect hand gestures. Mediapipe uses a single shot palm detection, and once that process is done, it performs a precise key point localization of 21 palm coordinates in the detected hand.

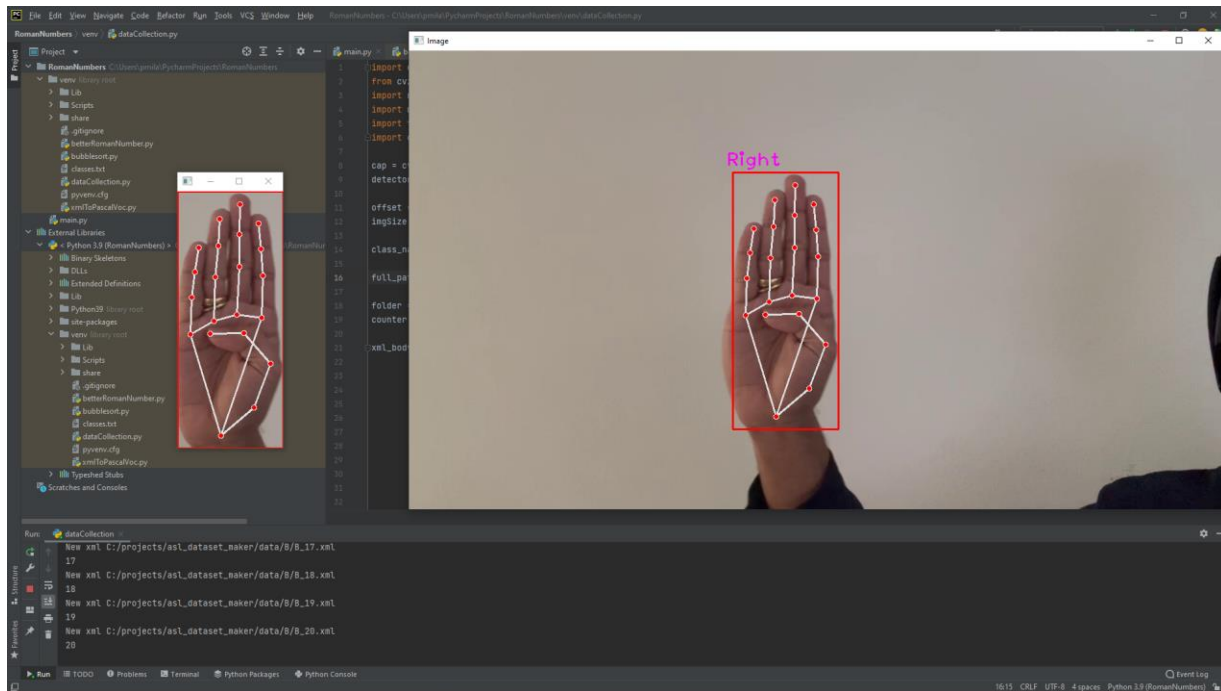


Figure 7: Hand Detector Module by CVZone for dataset acquisition

Overall, given the 3 stages of data collection for the dataset(Initially using the labelImg tool to manually label the images, then using a python script to label the dataset, and finally making some upgrades to the previous python script for data collection), It took around 2 months to have a dataset ready which can be used to train the machine learning models, since it required a good amount of research with what kind of datasets were required for different machine learning models.

```

<annotation>
  <folder>FOLDER</folder>
  <filename>A_0.jpg</filename>
  <path>C:/projects/asl_dataset_maker/data/A/A_0.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1280</width>
    <height>720</height>
    <depth>3</depth>
  </size>
  <object>
    <name>A</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>429</xmin>
      <ymin>255</ymin>
      <xmax>625</xmax>
      <ymax>457</ymax>
    </bndbox>
  </object>
</annotation>

```

Figure 8: XML File Example

The above figure shows an example of an XML file, this XML file contains all the information about the specific image and their respective bounding boxes. Information which is readable by the machine learning model and by humans too, in order for the training process. As shown in the figure, a typical XML file consists of information such as:

- Filename(in the above example, it is for the letter A_0, the very first image)
- Image path(where the image is stored)
- Image size(in this case, 1280x720), and
- The bounding box co-ordinates on that image

Chapter 3: Performance Metrics

To measure the success rate of the model, the best metric was the precise prediction of how accurately the model can detect the specific letter from American Sign Language(ASL).

True Positive(TP): A test outcome that accurately identifies the existence of a condition or trait. These are cases where the model predicts the ASL Letter and the original outcome in the dataset is the same.

True Negative(TN): A test outcome that accurately demonstrates the absence of a condition or trait. These are cases where the model predicts the wrong ASL Letter and the original outcome in the dataset is the same.

False Positive(FP): A test result that falsely suggests the presence of a certain condition or quality. These are cases where the model predicts the ASL letter and the original outcome in the dataset is that the predicted letter is wrong.

False Negative(FN): A test result that falsely suggests the absence of a certain ailment or quality. These are the cases where the model predicts the wrong ASL letter and the original outcome in the dataset is that the predicted letter is not wrong.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 9: Confusion Matrix[7]

	SSD Mobnet V2	Faster R-CNN	YOLO V8
Accuracy(%)	81	78	95
Average Recall(%)	0.84	0.843	0.93
Objectness Loss(%)	0.06	0.04	0.13

The above table displays the different performance metrics of the 3 machine learning models used in this research.

One parameter for assessing classification models is accuracy. The percentage of predictions that our model correctly predicted is known as accuracy.

The true positive rate (TPR), also known as recall, is the proportion of data samples from a class of interest—the "positive class"—that a machine learning model correctly recognizes as belonging to the class.

The likelihood that an object is present in a suggested region of interest is gauged by objectness loss. If the objectness is high, an object is probably present in the image window.

Chapter 4: Methodology

4.1 Machine Learning Models

There are various machine learning models that can be trained for object detection, and in this case, for accurately detecting and interpreting the American Sign Language(ASL). Different machine learning models such as

Faster R-CNN:

Faster R-CNN is a region-based model which utilizes a convolutional neural network(CNN) to showcase regions of interest and then classifies those regions into different categories.

Other types of R-CNN are Region-based Convolutional Neural Networks(R-CNN),

Fast R-CNN

YOLO:

YOLO stands for “you only look once”, as the name itself suggests, YOLO is a single shot detection model which uses a neural network to directly predict the bounding boxes and class labels for different objects in the image.

SSD:

SSD stands for “Single Shot Detector”, this is another single shot detection model just like the YOLO, which also uses neural networks to directly predict the bounding boxes and predict the class labels for different types of objects in the image.

RetinaNet:

RetinaNet is a variant of the R-CNN model which utilizes the focal loss function to address the class imbalances problem in detecting an object. Focal loss modifies the cross entropy loss by adding a term that adjusts emphasis on misclassified examples, with the main aim being to improve learning in those cases.

Mask R-CNN:

Mask R-CNN is an extension of the Faster R-CNN model, and branches out where the object detection is done in 2 ways/steps, Mask R-CNN predicts object masks in addition to bounding boxes and class labels/probabilities like YOLO and SSD.

Cascade R-CNN:

Cascade R-CNN is also another variant of the Faster R-CNN model which utilizes cascade detectors to improve the accuracy at which the model detects objects.

And other machine learning models can be used for object detection such as

- Region-based Fully Convolutional Network (R-FCN)
- Histogram of Oriented Gradients (HOG)
- Spatial Pyramid Pooling (SPP-net)
- Blitznet

4.2 Models Used

The 3 models which I have used for this research are SSD MobilenetV2, Faster R-CNN, and Yolo V8. The motive behind choosing these 3 models was because all of them have their own advantages and disadvantages over others. YOLO is the fastest for object detection as it analyses pictures at a real-time rate of 45 frames per second while SSD is more accurate in results than YOLO, since SSD can handle images that contain small objects, such as birds pretty well compared to YOLO.

Faster R-CNN is both, fast and accurate the only drawback being that it requires a huge amount of dataset to provide a decent amount of accuracy.

4.2.1 SSD Mobilenet V2

SSD MobileNet V2 is a popular object detection model which combines 2 deep learning architectures, MobileNet V2 and SSD(which stands for single shot detector). Mobilenet V2 is a lightweight CNN(Convolutional Neural Network) that is designed for better performance in mobile devices having limited computational resources. MobileNet V2 is known for its speed, efficiency, and high accuracy for object detection. SSD is an object detection framework that can detect multiple objects in an image with high accuracy and efficiency.

SSD combined with MobilNet V2 is a really strong combination as it creates a very powerful object detection model which is optimized for mobile devices. One of the major advantages of using the SSD MobileNet V2 is that it is designed to detect objects in real-time, which makes it suitable for this research and other areas such as autonomous driving, surveillance systems, and robotics.

SSD MobileNet V2 is available in a lot of different pre-trained models which can be used for different kinds of object detection, as well as, you can train these pre-trained models over your own dataset. In this research, I have used the SSD MobileNet V2 which is a pre-trained model and trained it over my own custom dataset which consists of over 6000 images with a total of 26 labels(for each alphabet in the English Language).

Different types of SSD MobileNet V2 are, SSD MobileNet V2 320x320, 512x512 and 640x60. All of these have their own advantages and disadvantages such as the smaller versions like SSD

MobileNet V2 320x320 are faster and more efficient and can be used for small use cases for faster outputs, on the other hand larger versions such as SSD MobileNet V2 640x640 are used where the use cases require higher accuracies.

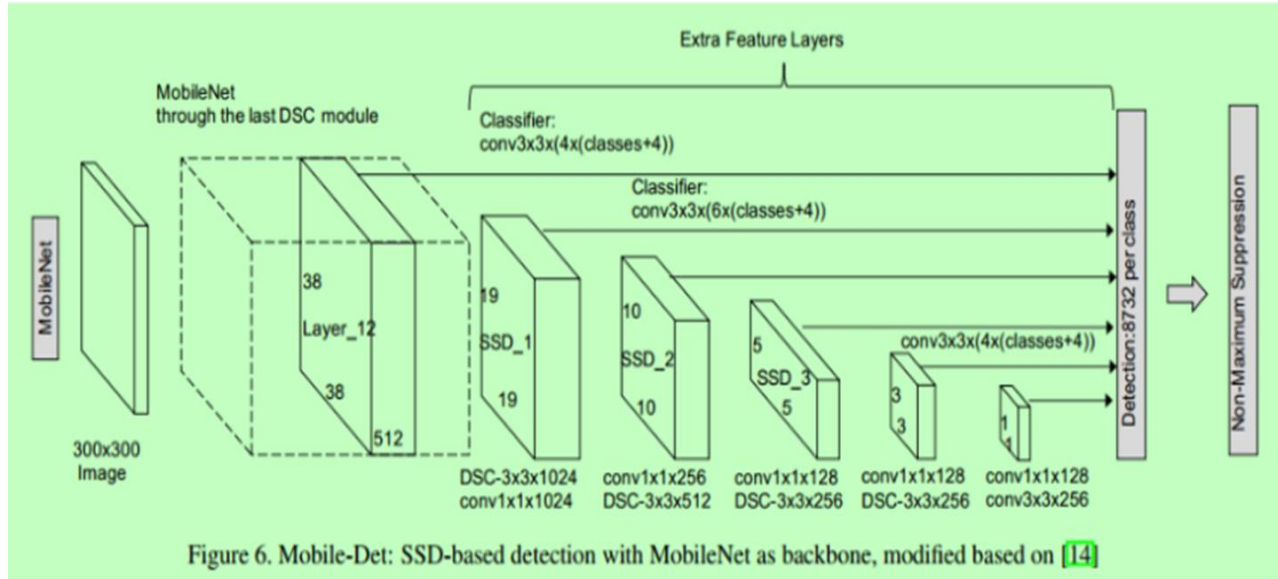


Figure 10: SSD MobileNet V2 architecture[8]

The SSD MobileNet V2 architecture consists of two main components: a MobileNet V2 backbone network and a Single Shot Detector (SSD) object detection network.

The MobileNet V2 backbone network is a lightweight convolutional neural network that is designed to perform well on mobile devices with limited computational resources. The 53 convolutional layers that make up the backbone network include depthwise separable convolutions, allowing for effective feature extraction with fewer parameters. The input image is utilized to extract features from the MobileNet V2 backbone network, which has been pre-trained on a sizable dataset of photos.

The SSD object detection network is used to detect the objects in the image by predicting the bounding boxes and class labels for each object. In this case, Label maps were created to make the model map between the alphabets, for example: For the alphabet “A”, Id was initialized 0, so when the model is trained with the images of A and the test is carried out, model maps to the said ID in this instance “0” hence that's how the model knows how to classify images for all sets of alphabets.

The SSD network has a number of detection layers that can identify objects with various scales and aspect ratios. The MobileNet V2 backbone network's feature extraction layers are connected to the detection layers, enabling them to extract features from the input image.

The SSD network consists of multiple layers, some of them being

Convolutional Layer: The convolutional layers are used to extract features from the input images.

Activation Layer: The activation layers are used to introduce non-linearity into the network.

Prediction Layer: The prediction layers are used to predict the class labels and bounding box coordinates for each image in that object.

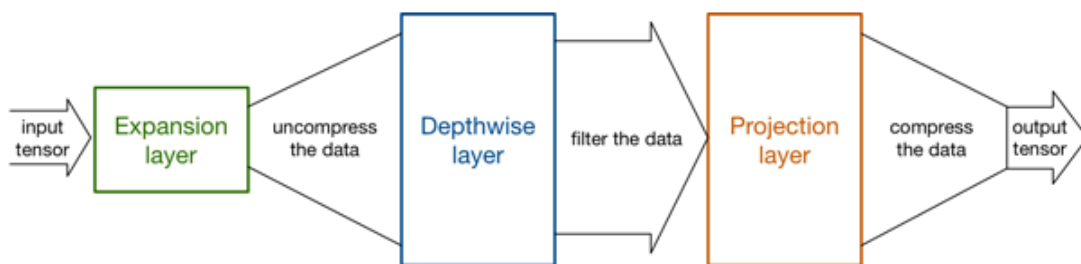


Figure 11: SSD MobileNet V2 layers [5]

4.2.1.i) Results from SSD MobileNet V2

SSD MobileNet V2 was trained 3 times over the same dataset with varying batch sizes and a number of steps it was trained for to get better accuracies. The training of the model was done using SSD MobileNet which is a pre-trained object detection model. The batch size of the data was 2 and The number of steps where the model started showing 0 learning rate was at the 61,000 Step with a total loss of 0.24 and a precision of 80%.

This training was done with a train-test split of approximately 80-20% with a dataset consisting of approximately 3000 Images. Label maps were created to make the model map between the alphabets, for example: For the alphabet “A”, Id was given 0, so when the model is trained with the images of A and the test is carried it, the model maps to the said ID in this instance “1” hence that's how the model knows how to classify images for all sets of alphabets.

```
labels = [{'name':'A', 'id':0}, {'name':'B', 'id':1}, {'name':'C', 'id':2}, {'name':'D', 'id':3}, {'name':'E', 'id':4},
          {'name':'F', 'id':5}, {'name':'G', 'id':6}, {'name':'H', 'id':7}, {'name':'I', 'id':8}, {'name':'J', 'id':9},
          {'name':'K', 'id':10}, {'name':'L', 'id':11}, {'name':'M', 'id':12}, {'name':'N', 'id':13}, {'name':'O', 'id':14},
          {'name':'P', 'id':15}, {'name':'Q', 'id':16}, {'name':'R', 'id':17}, {'name':'S', 'id':18}, {'name':'T', 'id':19},
          {'name':'U', 'id':20}, {'name':'V', 'id':21}, {'name':'W', 'id':22}, {'name':'X', 'id':23}, {'name':'Y', 'id':24},
          {'name':'Z', 'id':25}, {'name':'hello', 'id':26}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{ }\'\n'.format(label['name']))
        f.write('\tid:{ }\n'.format(label['id']))
        f.write('}\n')
```

Figure 12: Label Map

Below is the picture of training the model in step 1 with 61,000 steps, having a 80% accuracy and 0.24 Loss:

```
creating index...
index created!
INFO:tensorflow:Loading and preparing annotation results...
I1127 13:15:13.883668 4076 coco_tools.py:116] Loading and preparing annotation results...
INFO:tensorflow:DONE (t=0.01s)
I1127 13:15:13.893667 4076 coco_tools.py:138] DONE (t=0.01s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.97s).
Accumulating evaluation results...
DONE (t=0.38s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.809
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.989
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.922
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.800
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.814
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.844
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.845
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.845
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.800
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.849
INFO:tensorflow:Eval metrics at step 61000
I1127 13:15:15.292681 4076 model_lib_v2.py:1007] Eval metrics at step 61000
INFO:tensorflow: + DetectionBoxes_Precision/mAP: 0.808874
I1127 13:15:15.299680 4076 model_lib_v2.py:1010] + DetectionBoxes_Precision/mAP: 0.808874
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.50IOU: 0.988779
```

Figure 13: SSD MobileNet V2 Training

Results of training the SSD MobileNet V2 for the same dataset for 61,000 steps is shown below:

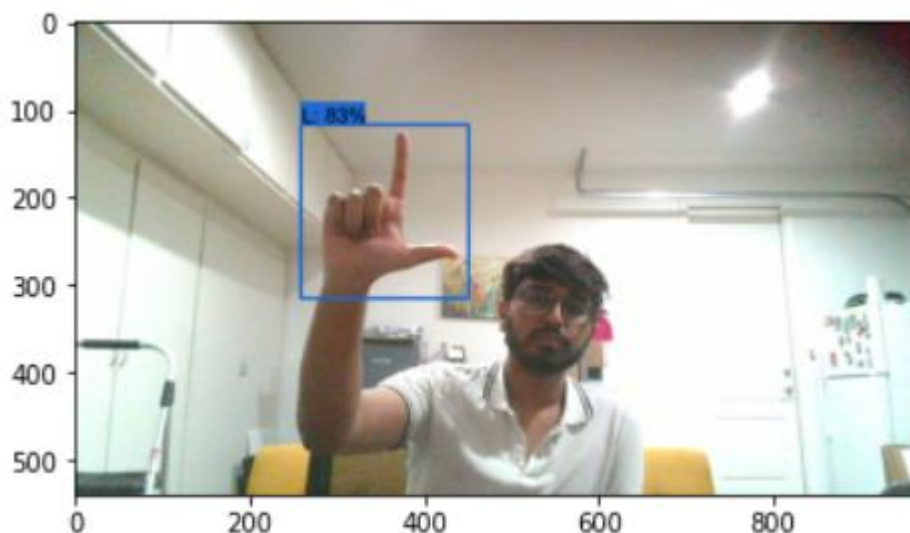
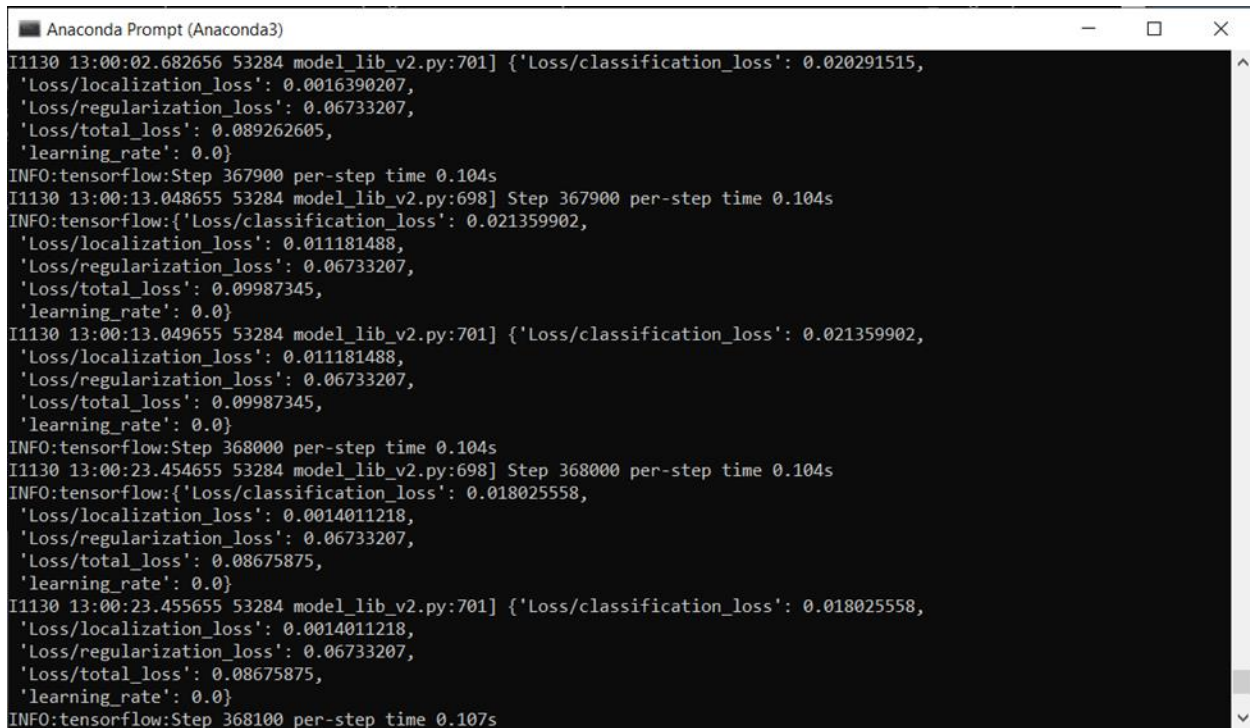


Figure 14: SSD MobilNet V2 Result

The training for the second model, the same pipeline config was used, Batchsize of 2 And the number of steps were 1000000, although, after step number approximately 300000 steps, the learning rate of the model became 0 with a Loss of 0.06.



```
Anaconda Prompt (Anaconda3)
I1130 13:00:02.682656 53284 model_lib_v2.py:701] {'Loss/classification_loss': 0.020291515,
'Loss/localization_loss': 0.0016390207,
'Loss/regularization_loss': 0.06733207,
'Loss/total_loss': 0.089262605,
'learning_rate': 0.0}
INFO:tensorflow:Step 367900 per-step time 0.104s
I1130 13:00:13.048655 53284 model_lib_v2.py:698] Step 367900 per-step time 0.104s
INFO:tensorflow: {'Loss/classification_loss': 0.021359902,
'Loss/localization_loss': 0.011181488,
'Loss/regularization_loss': 0.06733207,
'Loss/total_loss': 0.09987345,
'learning_rate': 0.0}
I1130 13:00:13.049655 53284 model_lib_v2.py:701] {'Loss/classification_loss': 0.021359902,
'Loss/localization_loss': 0.011181488,
'Loss/regularization_loss': 0.06733207,
'Loss/total_loss': 0.09987345,
'learning_rate': 0.0}
INFO:tensorflow:Step 368000 per-step time 0.104s
I1130 13:00:23.454655 53284 model_lib_v2.py:698] Step 368000 per-step time 0.104s
INFO:tensorflow: {'Loss/classification_loss': 0.018025558,
'Loss/localization_loss': 0.0014011218,
'Loss/regularization_loss': 0.06733207,
'Loss/total_loss': 0.08675875,
'learning_rate': 0.0}
I1130 13:00:23.455655 53284 model_lib_v2.py:701] {'Loss/classification_loss': 0.018025558,
'Loss/localization_loss': 0.0014011218,
'Loss/regularization_loss': 0.06733207,
'Loss/total_loss': 0.08675875,
'learning_rate': 0.0}
INFO:tensorflow:Step 368100 per-step time 0.107s
```

Figure 15: SSD Mobilenet V2 training iteration 2

Results of training the SSD MobileNet V2 for the same dataset for 300000 steps is shown below:

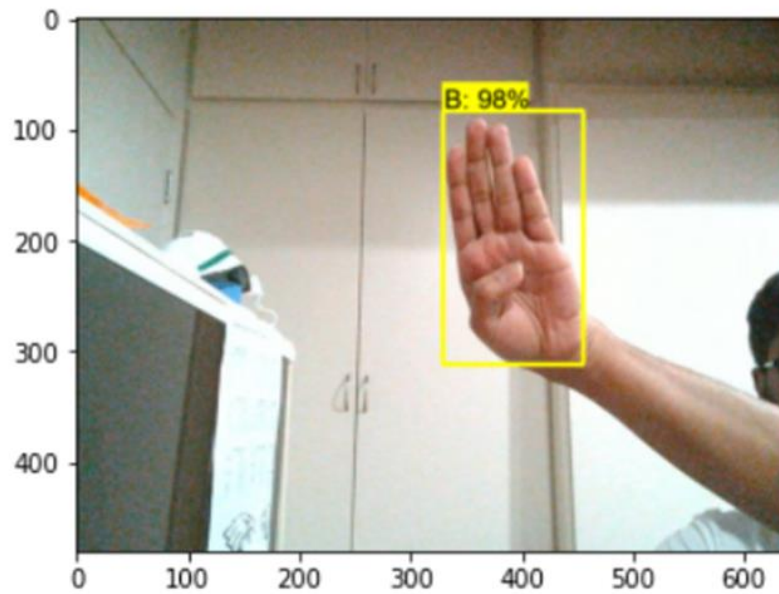


Figure 16: SSD MobileNet V2 Result

4.2.2 YOLO(You Only Look Once)

YOLO(You only look once) is a real-time object detection system that utilizes deep neural networks to detect objects from video frames and images. In order to estimate bounding boxes and class probabilities for each cell, YOLO divides each picture into a grid of cells. Four coordinates (x, y, w, h) are used to represent each bounding box, where (x, y) stands for the box's center and (w, h) for its width and height. The class probabilities show the possibility that a specific class of object is contained in the box.

The YOLO model consists of a convolutional neural network(also known as CNN) that is used to extract features from images that are used as input, followed by connected layers that predict the bounding boxes and class labels/probabilities. The convolutional neural network(CNN) used in YOLO is based on Darknet architecture, which is a smaller and faster version of VGG and ResNet architecture.

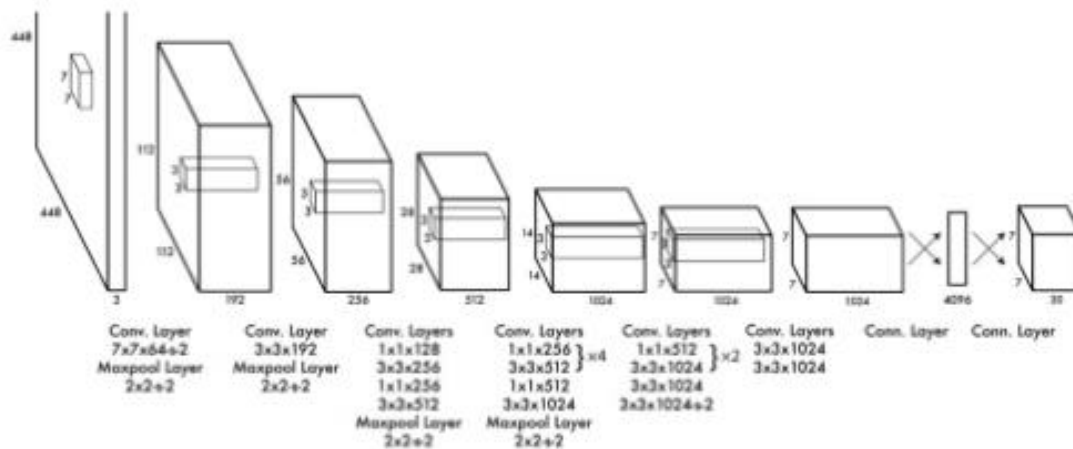


Figure 17: YOLO Architecture [9]

YOLO employs a loss function during training that penalizes inaccurate predictions of the bounding box coordinates as well as the class probabilities. The localization loss, confidence loss, and classification loss are the three terms that make up the loss function. When comparing predicted and actual bounding box coordinates, the localization loss measures the difference; when comparing expected and actual class probabilities, the confidence loss measures the difference. The classification loss penalizes the model when it predicts an object's class erroneously.

One of the advantages of using YOLO is its speed and accuracy. As the name itself suggests, YOLO(You Only Look Once), YOLO can achieve high speeds and accuracy in detecting objects from video frames and images because it only needs to make one pass through the network to predict the bounding box and class labels/probabilities. Additionally, real-time object detection can be achieved by YOLO on a CPU.

4.2.2.1 Training YOLO model:

Training a YOLO model for object detection involves several steps:

Data Collection: Gathering and labeling a dataset of photos with the things you wish to detect is the first stage. This dataset ought to be sufficiently varied to account for various lighting setups, scales, and viewpoints.

Data Preparation: After gathering the dataset, you must get it ready for training. The photographs must be reduced to a set size, converted to the proper format (such as JPEG or PNG), and annotated with files describing the locations and types of the items in each image.

Unlike Faster R-CNN and SSD MobileNet V2, for annotations, I used PascalVoc files instead of xml to define the annotations and bounding boxes boundaries.

Example of an XML and PascalVoc File are as shown below:

```
<annotation>
  <folder>FOLDER</folder>
  <filename>A_0.jpg</filename>
  <path>C:/projects/asl_dataset_maker/data/A/A_0.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1280</width>
    <height>720</height>
    <depth>3</depth>
  </size>
  <object>
    <name>A</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>429</xmin>
      <ymin>255</ymin>
      <xmax>625</xmax>
      <ymax>457</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 18: XML File Example

```
0 0.41171875 0.49444444444444446 0.153125 0.28055555555555556
```

Figure 19: Pascal VOC file Example

Since the same dataset was utilized for all 3 models, I wrote a script in Python to convert the already gathered images and their respective XML files to PascalVoc.

Visual Object Classes, or Pascal VOC, is a well-known benchmark and dataset for object detection and segmentation in computer vision. Since its initial introduction in 2005, it has been frequently utilized as a benchmark in the computer vision industry.

Images of 20 various item categories, including people, cars, and animals, can be found in the Pascal VOC dataset. Also, each image has annotations that describe the location and type of each

4.2.3 Faster R-CNN

Modern object detection algorithms, such as Faster R-CNN, were unveiled in 2015 by Ross Girshick et al.[3] To increase the detection accuracy and speed, it draws upon the R-CNN (Region-based Convolutional Neural Networks) and Fast R-CNN algorithms.

Faster R-CNN's Region Proposal Network (RPN) is the initial stage in the process and it generates potential object areas or bounding boxes. After receiving an input image, the RPN generates a number of object recommendations, each with a corresponding objectness score.

The second stage of Faster R-CNN uses the Region of Interest (RoI) Pooling layer to extract a fixed-size feature map from each proposal region. The object is then classified and its bounding box is predicted using this feature map, which is subsequently input into a fully connected layer.

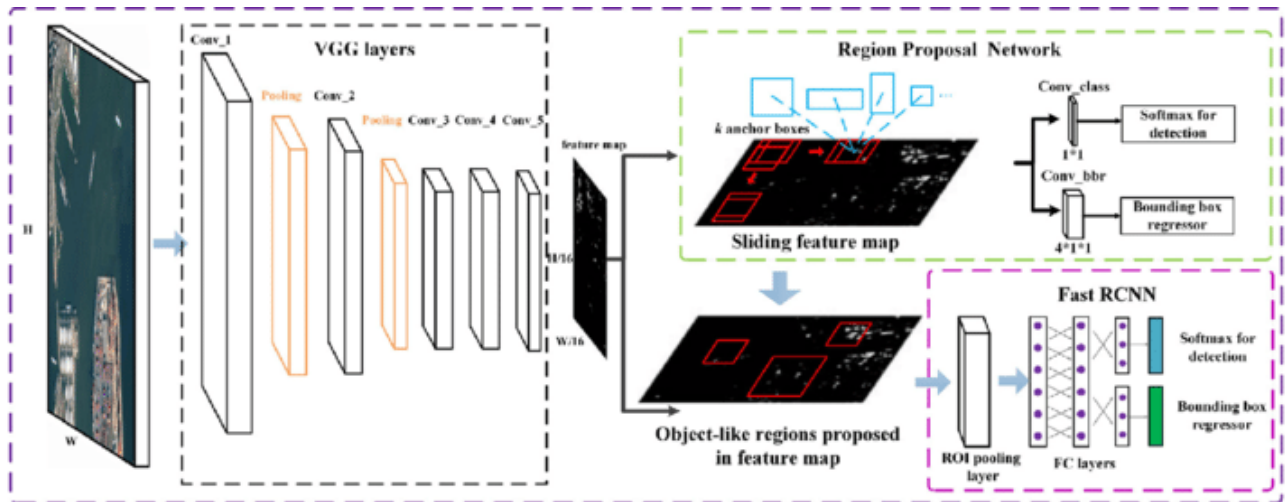


Figure 22: Faster R-CNN Architecture [10]

Training: Using a multi-task loss function that integrates the classification and regression losses, the Faster R-CNN is trained from beginning to end. The accuracy of object classification is measured by the classification loss, and the accuracy of bounding box regression is measured by the regression loss.

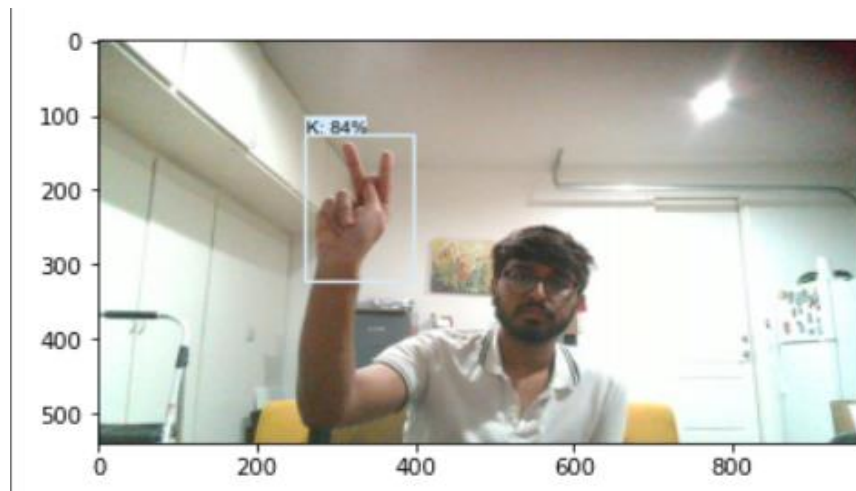


Figure 23: Faster R-CNN Results

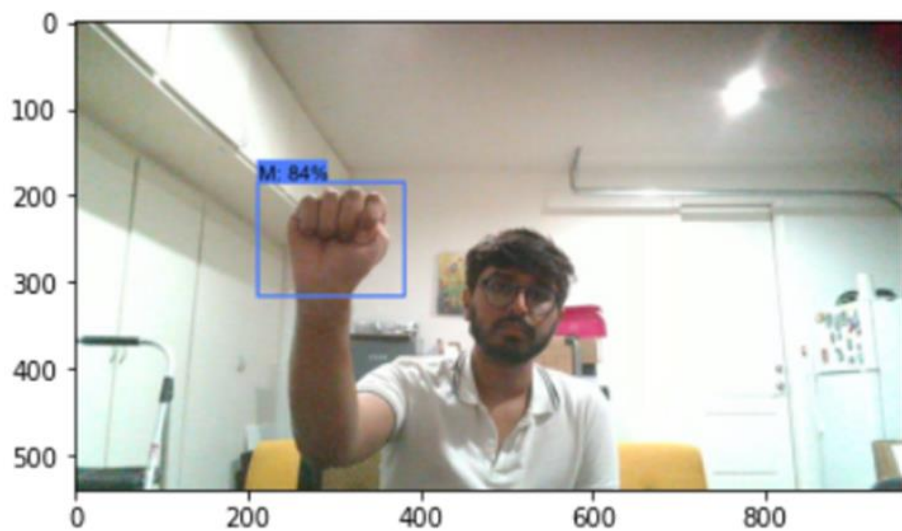
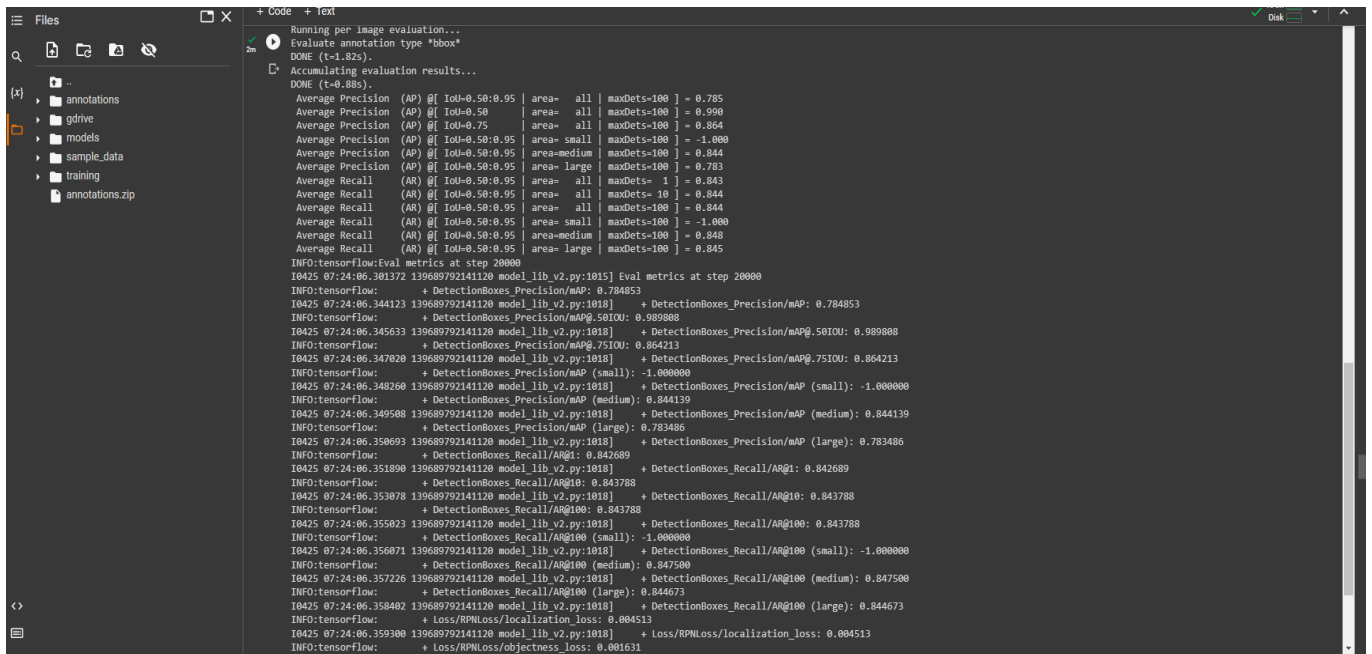


Figure 24: Faster R-CNN Results

4.2.3.1 Faster R-CNN Results:



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'annotations', 'gdrive', 'models', 'sample_data', 'training', and 'annotations.zip'. The code editor displays the output of a 'Running per image evaluation...' command. The output includes a table of evaluation metrics for different IoU thresholds and area sizes, followed by a series of INFO messages showing the training progress and final metrics.

```
Running per image evaluation...
Evaluate annotation type 'bbox'
DONE (t=1.82s).
Accumulating evaluation results...
DONE (t=0.88s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.785
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.990
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.864
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area= medium | maxDets=100 ] = 0.844
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.783
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.843
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.844
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.844
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= medium | maxDets=100 ] = 0.848
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.845
INFO:tensorflow:Eval metrics at step 20000
10425 07:24:06.381372 139689792141120 model_lib_v2.py:1015] Eval metrics at step 20000
INFO:tensorflow: + DetectionBoxes_Precision/mAP: 0.784853
10425 07:24:06.344123 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP: 0.784853
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.50IOU: 0.989808
10425 07:24:06.345633 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.50IOU: 0.989808
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.75IOU: 0.864213
10425 07:24:06.347020 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.75IOU: 0.864213
INFO:tensorflow: + DetectionBoxes_Precision/mAP (small): -1.000000
10425 07:24:06.348260 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP (medium): 0.844139
10425 07:24:06.349588 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (medium): 0.844139
INFO:tensorflow: + DetectionBoxes_Precision/mAP (large): 0.783486
10425 07:24:06.350693 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (large): 0.783486
INFO:tensorflow: + DetectionBoxes_Recall/AR@1: 0.842689
10425 07:24:06.351890 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@1: 0.842689
INFO:tensorflow: + DetectionBoxes_Recall/AR@10: 0.843788
10425 07:24:06.353078 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@10: 0.843788
INFO:tensorflow: + DetectionBoxes_Recall/AR@100: 0.843788
10425 07:24:06.355023 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100: 0.843788
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (small): -1.000000
10425 07:24:06.356071 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (medium): 0.847500
10425 07:24:06.357226 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (medium): 0.847500
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (large): 0.844673
10425 07:24:06.358482 139689792141120 model_lib_v2.py:1018] + DetectionBoxes_Recall/AR@100 (large): 0.844673
INFO:tensorflow: + Loss/RPNLoss/Localization_loss: 0.804513
10425 07:24:06.359380 139689792141120 model_lib_v2.py:1018] + Loss/RPNLoss/Localization_loss: 0.804513
INFO:tensorflow: + Loss/RPNLoss/objectness_loss: 0.001631
```

Figure 25: Faster R-CNN Results

As the above figure shows the training steps, training parameters and the results from the Faster R-CNN model, this model achieved an accuracy of 78%, when it was trained over the same dataset, with a batch size of 8, and the number of training steps being 20,000 steps. The average recall rate being 0.843 and having an objectness loss of 0.04.

4.3 Training Parameters

Other than dataset acquisition, another topic in training the models was researching, tweaking and setting up the appropriate training parameters. Training parameters such as:

- Knowledge of what kind of files the models use(XML files/Pascal VOC)
- Having the proper balance of setting the batch size
- OOM(Out of memory) error prevention
- Making sure the models don't overfit
- Having the knowledge of when to stop training the machine learning models so that you don't use unnecessary resources(That is, after a certain point, when the learning rate of the machine learning model becomes 0, there is no point in training the machine learning model for more epochs/steps since it won't be learning anything from the dataset).
- Major one being that, utilizing the NVIDIA GPU to train the models, which was way faster and more efficient than compared to training on your CPU which pretty much takes double the time to train the models. The advantages being:
 - Saves time
 - Saves resources
 - Less load on your machine
 - More efficient

	SSD Mobilenet V2	Faster R-CNN	YOLO V8
Batch Size	4	8	4
No. Of Classes	26	26	26
Steps/Epochs	100000 Steps	20000 Steps	100 epochs
CPU/GPU	NVIDIA GPU	TESLA T4	TESLA T4

Batch Size: The number of samples that are processed before the model is changed is the batch size. The quantity of complete iterations through the training dataset is the number of epochs. The minimum and maximum sizes for batches are one and the number of samples in the training dataset, respectively.

No. Of Classes: In this case, it means the different types of dataset labels used(26 for the 26 letters in the Alphabets from A-Z).

Steps/Epochs: One gradient update is one training step. Examples of batch_size are processed in a single step. One complete cycle of the training data constitutes an epoch. This normally involves several processes.

Chapter 5: Comparing the Results

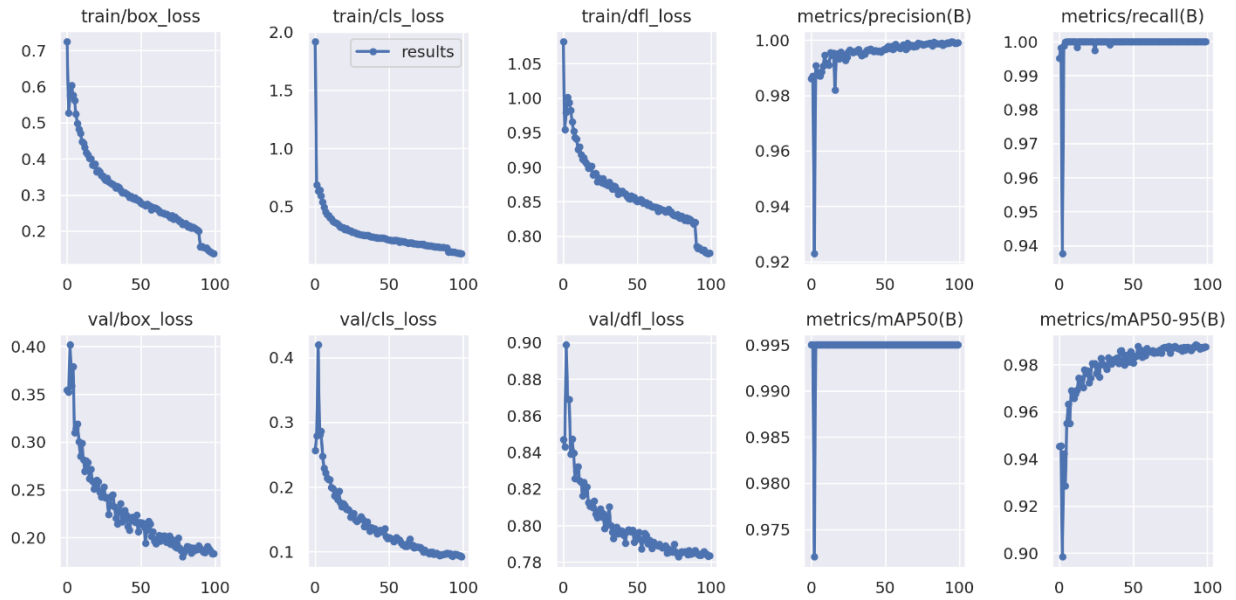


Figure 26: Performance Metrics of the Models

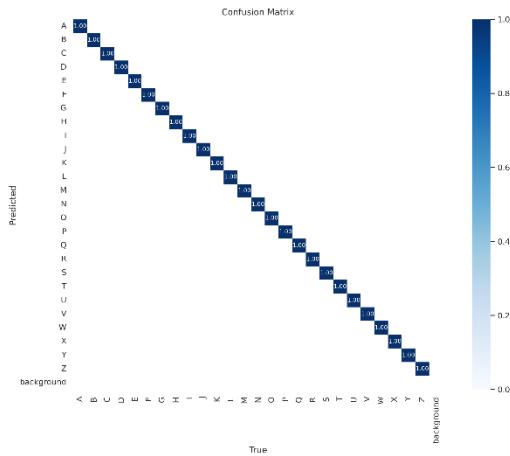


Figure 27: Confusion Matrix

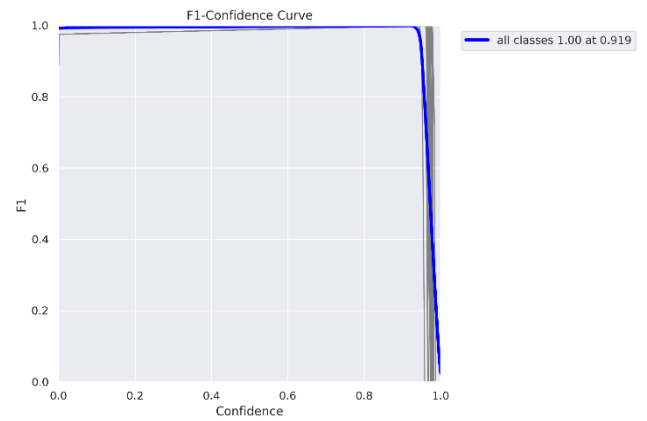


Figure 28: F-1 Confidence Curve

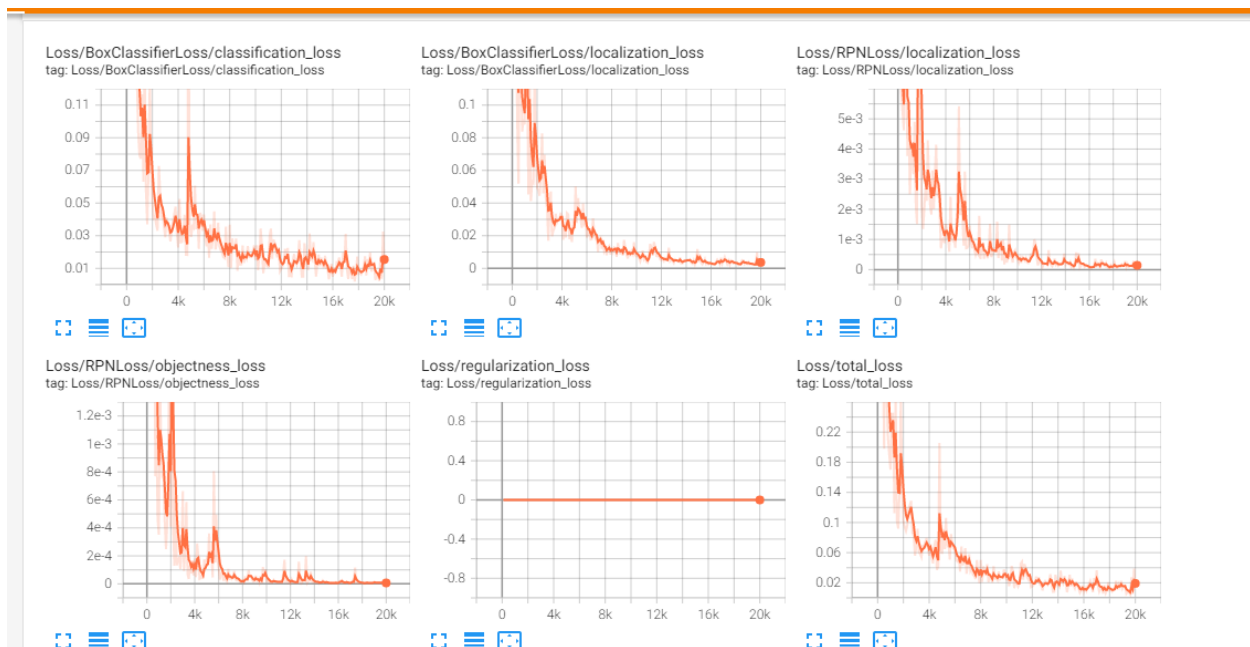


Figure 29: Training graphs of Loss of models

learning_rate

learning_rate
tag: learning_rate

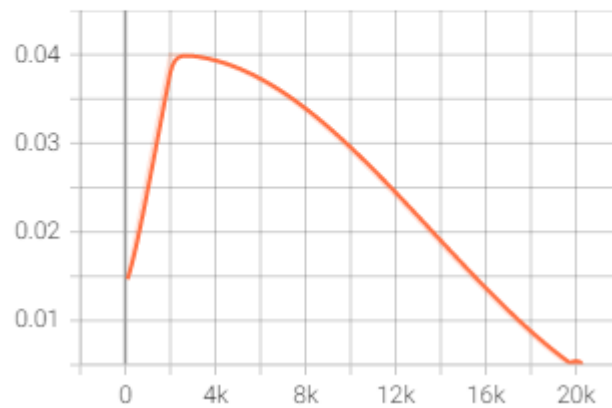


Figure: 30 Training graphs of Loss of models

The above figures displays different training results/metrics obtained once the models are trained, metrics such as box loss, class loss, confusion matrix, learning rate.

Box Loss: The box loss measures how accurately the algorithm can pinpoint an object's center and how completely the anticipated bounding box encloses an object.

Classification Loss: Loss functions for classification are computationally viable loss functions that quantify the cost of inaccurate predictions in classification issues (problems of determining which group a specific observation belongs to). These loss functions are used in machine learning and mathematical optimization.

Total Loss: The consequence of a poor prediction is loss. In other words, loss is a measure of how poorly the model predicted a single case. The loss is zero if the model's forecast is accurate; otherwise, the loss is higher.

Confusion Matrix: A confusion matrix aids in visualizing the results of a classification task by providing a table arrangement of the various outcomes of the prediction and findings. It creates a table with all of a classifier's predicted and actual values.

Learning Rate: A hyper-parameter used to control how quickly an algorithm updates or learns the values of a parameter estimate is known as the learning rate and is represented by the symbol. In other words, our neural network's weights with respect to the loss gradient are controlled by the learning rate.

Chapter 6: Summary

6.1 Summary:

In summary, the SSD Mobilenet V2 architecture combines the efficiency of MobileNet V2 with the accuracy of SSD(Single Shot Detector) to create a powerful, efficient and accurate object detection model that is optimized for mobile devices. The architecture consists of the MobileNet V2 backbone network, the SSD object detection network and anchor boxes for generating bounding boxes around object locations.

In general, Faster R-CNN is an effective object detection algorithm that builds on the advantages of earlier methods while also introducing a number of significant advancements to increase accuracy and speed.

Careful data preparation, model configuration, and training are required when training a YOLO model for object identification. However, a highly precise and effective object detection system can be created with the appropriate methods and tools.

6.2 Conclusion:

As seen in the table and results discussed in previous sections of the research paper and comparing the different performance metrics, statistically the YOLO V8 was better, in terms of accuracy and recall, however a firm conclusion cannot be drawn from just numbers as we have to take into account of how the models perform in real life scenarios too. If we compare the actual results and output from the models, we can conclude that the SSD MobileNet V2 model was better,

faster and accurate and could accurately detect the sign language in real life scenarios. Another advantage of the SSD MobileNet V2 being that it can be used on small devices such as a mobile phone too, it is fast, accurate, provides real time feedback and works well across small to large scale implementations.

6.3 Learning Experience

This project development was a big learning curve for me in general where I was exposed to many new learning which helped me hone my skills in the machine learning field. I learned to work with notebooks/cloud based notebooks such as Jupyter notebook and Google Colab, getting accustomed to different python libraries, creating your own dataset, learn to make your own python scripts to make the work easier, and learn about different machine learning concepts and getting hands on experience by successfully building 3 different machine learning models

6.4 Future Work

As this is the first step to working towards an interface and an application, which could serve as a bridge of communication between people who know sign language and people who don't, many things can be built on top of this foundation such as:

- Use many other machine learning models to further branch out and explore about their performance and accuracies, models such as VGG16, Fast R-CNN and other CNN models.
- Utilize bigger datasets for better accuracies.
- Create an App or an interface.
- Make an application which not only detects sign language letters in real time, but also sign language words.

References

- [1] https://en.wikipedia.org/wiki/American_School_for_the_Deaf
- [2] <https://www.startasl.com/history-of-sign-language/>
- [3] <http://www.cdhh.ri.gov/information-referral/american-sign-language.php>
- [4] <https://ieeexplore.ieee.org/ielam/34/7346524/7112511-aam.pdf>
- [5] <https://machinethink.net/blog/mobilenet-v2/>
- [6] <https://www.gerardaflaguecollection.com/products/american-sign-language-asl-alphabet-abc-poster.html>
- [7] <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>
- [8] <https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd0691>
- [9] <https://www.v7labs.com/blog/yolo-object-detection>
- [10] https://www.researchgate.net/figure/The-architecture-of-Faster-R-CNN_fig2_324903264