SIGN LANGUAGE RECOGNITION USING PYTHON AND OPENCV

A Project Report

Submitted in partial fulfilment of the requirements

for the award of the degree of

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

By

Shlok G. Shivkar

Seat Number :- 21306A1045

Vaishnavi P. Pangam

Seat Number :- 21306A1074

Under the esteemed guidance of

Mrs. Pallavi Tawade

Assistant Professor, Department of Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

VIDYALANKAR SCHOOL OF INFORMATION TECHNOLOGY

(Affiliated to University of Mumbai)

MUMBAI, 400 037

MAHARASHTRA

2022 - 2023

VIDYALANKAR SCHOOL OF INFORMATION TECHNOLOGY

(Affiliated to University of Mumbai)

MUMBAI-MAHARASHTRA-400037

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project entitled, **"Sign Language Recognition Using Python And OpenCV"**, is bona fide work of **Shlok Shivkar & Vaishnavi Pangam** bearing Seat No :- 21306A1045 & 21306A74 submitted in partial fulfilment of the requirements for the award of degree of MASTER OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide **Coordinator**

Internal Examiner **External Examiner**

Date: **College Seal** **Principal**

Research Paper

SIGN LANGUAGE RECOGNITION USING PYTHON AND OPENCV

Shlok G. Shivkar
Department Of Information Technology
Vidyalankar School of Information Technology
Vidyalankar College Road, Wadala (East), Mumbai, Maharashtra 400 037
Email: shlokshivkar21@gmail.com
Mobile: 8369621421

Vaishnavi Pangam
Department Of Information Technology
Vidyalankar School of Information Technology
VidyalankarCollegeRoad, Wadala (East), Mumbai, Maharashtra 400 037
Email: vaishnavipangam.vp@gmail.com
Mobile: 9136541025

1. Abstract

This study examines the various stages of an automatic system for the identification of sign language (SLR). A large dataset and the best algorithms must be used to train a system that can read and understand a sign. In current society, there is a lack of communication with the deaf. The use of Sign Language (SL) helped to break down this barrier. Sign language uses visually conveyed sign patterns to communicate meaning to non-sign language users. It is also useful in communicating with individuals suffering from autism spectrum disorder (ASD). Normal people cannot understand the signs used by deaf people, as they do not know the meaning of a particular sign. The system presented is intended to address this issue. The system captures various gestures of the hand by using a webcam. Pre-processing of the image takes place, then, determination of edges occurs by using object detection. Finally, a template-matching algorithm identifies the sign and displays the text. The output is in textual format so one can easily interpret the meaning of a particular sign. This also curtails the difficulty of communicating with the deaf. The implementation of the system is by creating libraries then by using TensorFlow Object Detection Pipeline Configuration for object detection and finally running the model in real time by using OpenCV in Python in real time and obtaining an audio message of the indicated hand sign.

Keywords: Deep Learning, Convolutional Neural Network, Recognition, Image Processing, Sign Language, Object Detection, Image Detection, Accuracy

2. Introduction

This project can be very helpful for deaf and dumb people communicating with others as knowing sign language recognition is not common to all. This can be extended to creating automatic creators where a person can easily write by hand gestures. Deep learning can be used to make an impact on this cause.

With this research, we hope we can build better communication with deaf people. Because of the complexity and diversity of Sign Language. It makes this topic even harder. Sometimes Sign Language has some similar sign with different meaning. Which is, it made Deep Learning harder to recognize them. And then we also should consider differences between people who do Sign Language.

Now, you can imagine how hard Deep Learning must learn many things about this topic to get a good accuracy and decent to be implemented as a device to interpret the Sign Language. Even using different dataset of the same method could make a significant difference of result. And improve this topic to be better.

Gesture analysis is a scientific field that can recognize gestures such as hand, arm, head, and even structural motions that usually entail a certain posture and/or motion. Using hand gestures, the individual may send out more information in a shorter amount of time. Several approaches were explored to apply computer-vision ideas to the real-time processing of gesture outputs. The Computer Vision study concentrates on gesture recognition in the open CV framework using the Python language. Language is a huge part in communication. Languages are useless to a person with a disability. Gesture is a vital and meaningful mode of communication for the visually impaired person. So here is the computer-based method for regular people to understand what the differently abled individual is trying to say. For monitoring, there are various similar algorithms and object recognition systems. This allows the identification of gestures, which overcomes the boundaries and limitations of earlier systems.

3. Review of Literature

Name of Topic	Author Names	Technology used	Algorithms used/Model used	Limitation
Sign language technique using template matching technique	Soma Shrenika Myneni Madhu Bala	Feature Extraction such as Canny edge detection, image smoothing and hysteresis process which converts weak pixels to strong pixels	Template matching algorithm	This model does not work in real-time, and we cannot have a live conversation with a differently abled person.
Data collection of 3D spatial features of gestures from static Peruvian sign language (PSL) alphabet for sign language recognition	Roberto Nurena-Jara Cristopher Ramos-Carrion Pedro Shiguihara-Juarez	HTC Vive device to extract 21 key points from hand to obtain feature vector	3D spatial positions of static gestures from the PSL alphabet	Too many instances in the model which makes it complex to solve.
Sign Language Recognition Using Modified Convolutional Neural Network Model	Suharjito Herman Gunawan Narada Thiracitta Ariadi Nugroho	Transfer learning with i3d, AI CNN	i3d inception is also known as a new action recognition model with high accuracy, CNN	Validation accuracy is low. This model is too overfit
Real Time Sign Language Recognition using PCA (Principal Component Analysis)	Shreyashi Narayan Sawant M S Kumbhar	Image processing (Recognizing 26 gestures)	Indian Sign Language by using MATLAB, pre-processing Hand segmentation Feature extraction, Sign recognition,	The gestures used in this model limits to only Indian sign language.

			Sign to text, Voice conversion using PCA algorithm	
Towards Sign language recognition system in Human-Computer interactions	Maher Jebali Patrice Dalle Mohamed Jemni	Semantic Analysis	Dialogue between deaf people Signing avatar prediction algorithms	Handling problems of large vocabulary complexity
American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning Features	Rajesh George Rajan Dr.M.Judith Leo	Deep Learning	Serial Based Feature Fusion and CNN Deep learning technology VGG-19	This model does not work in real-time, and we cannot have a live conversation with a differently abled person.
Attention based 3D-CNNs for Large-Vocabulary Sign Language Recognition	Jie Huang, Wengang Zhou Houqiang Li Weiping Li	Spatio-temporal feature learning	3D-CNN Feature Extraction RNN	-
Sign language recognition using image based hand gesture recognition techniques	Ashish S. Nikam Aarti G. Ambedkar	Image Preprocessing Image Enhancement Segmentation Color jiltering and skin segmentation	Slope distance-based algorithm Convexity hull algorithm	Less accurate than other conventional recognition methods
Dynamic Sign Language Recognition Based on Video	Yanqiu Liao,	Image recognition and Feature extraction	3D Residual ConvNet and Bi-directional LSTM networks	-

Based on our review, HMM-based approaches have been extensively explored in prior research, including its modifications. Deep Learning, such as Convolutional Neural Networks, has been popular in the past five years. Hybrid CNN-HMM and completely deep learning systems have yielded encouraging results and provide avenues for additional research. Clustering and high computational needs, however, continue to stymie their adoption. We believe that the research's future focus should be on developing a simplified network that can

reach high performance while requiring little CPU resources, and that embeds the feature learner within the classification in a multi layered neural network approach

- Normal People and Deaf-Dumb People Communication: -

The overall purpose of the project is facilitating the interaction between deaf and dumb people and normal people to makes the communication between normal people and dumb people easier, by translate the sign language to voice or text with high accuracy. The dumb and deaf communicate via sign language, which is hard to decipher for those who are not familiar with it. As a result, it is necessary to develop a device that can translate gestures into speech and text. This will be a significant step in allowing deaf and dumb people to communicate with the broader population.

- A System for Recognition of Indian Sign Language for Deaf People using Otsu's Algorithm:-

In proposed paper, some methods for making sign recognition easier for people while communicating and the result of those symbol signs will be converted into the text. In this project, we are capturing hand gestures through a webcam and converting this image into a grayscale image. The segmentation of the grayscale image of a hand gesture is performed using the Otsu thresholding algorithm.. The whole picture level is split into two categories: hand and backdrop.

The best threshold value is calculated by computing the proportion between total class variance and class variance. The Canny edge detection technique is used to locate the border of a hand gesture in a picture. We employed edge-based segmentation and threshold-based segmentation in Canny edge detection. Then Otsu's algorithm is used because of its simplified calculations and stability. This algorithm fails when the global distribution of the target and background varies widely.

- Image Processing for Intelligent Sign Language Recognition: -HMMs are suited for full sign recognition of ASL, because of their inherent time-varying nature. Because a series of several of the 36 basic hand shapes may be used to gesture most ASL signs. The continuous indications can be split, with the fundamental hand shapes retrieved as the input to the HMM processor. The fundamental hand shapes may then be identified and chained as ASL words' output. With the approaches presented in this work, the system may be expanded to a full-sign recognition system.
- Sign Language Interpreter using Machine Learning and Image Processing: -

Pham Microsoft Kinect is used by the Hai to interpret Vietnamese Sign Language. The user must align himself with Kinect's field of view and then conduct sign language movements in the suggested system. Using multiclass Support Vector Machine, it can distinguish both dynamic and static gestures. The gesture features are retrieved, filtered out and normalize on Euclidean distance during recognition

- American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning Features

In this paper, a skin color-based segmentation method is used to segment the images and these resultant segmented images are supplied to the proposed deep learning model as shown. In this

method a deep learning model called VGG-19 is used to extract the features. Afterward, the local features taken from the hand-crafted methods and deep features from deep neural net concatenated serially, finally a support vector machine (SVM) is used to classify the signs.

- Moment Based Sign Language Recognition for Indian Languages

Image Capture This is the first step in sign recognition Web camera is used to capture the hand gesture Image Preprocessing Image preprocessing contains cropping, filtering, brightness & contrast adjustment & many more. To do such process Image enhancement, Image cropping & Image segmentation methods are used. Feature Extraction Feature extraction is a very useful step to create the database of sign recognition

- Attention-Based 3D-CNNs for Large-Vocabulary Sign Language Recognition

2nd spatial attention-based 3D-CNNs to extract features from video. After that, we built a temporal attention-based model for classification

1. Spatial Attention based 3D-CNNs
2. Temporal Attention based Classifier

- Dynamic Sign Language Recognition Based on Video Sequence with BLSTM-3D Residual Networks

Dynamic sign language recognition method based on analysing the features of hand shapes and motion trajectory with each hand gesture is a normal solution to the problems of dynamic sign language recognition. Increased researchers have focused on sign language recognition based on video sequence. For the method of dynamic sign language recognition based on video sequence, sensors are not necessary elements.

5. Future Scope and Conclusion

In future work, the proposed system can be developed and implemented using raspberry PI. The image processing part should be improved so that system will be able to communicate in both directions I.e. It should be capable of converting normal language to sign language and vice-versa. It will try to recognize signs which include motion. Moreover, we will focus on converting the sequence of gestures into text I.e., words and sentences, and then converting it into speech which can be heard.

The aim of this paper is to help and serve the deaf of our society to communicate with normal people. Here the implementation of the system is using deep learning techniques. This system is for people who cannot use gloves, sensors, and other highly refined equipment. First, acquire images with a camera then run it through the specified algorithms under the created libraries and finally display the users performed hand sign, and produce an audio output depending on the detected sign.

6. References

- Ruchi Manish Gurav and Premanand K Kadbe. Real time finger tracking and contour detection for gesture recognition using opencv. In 2015 International Conference on Industrial Instrumentation and Control (ICIC), pages 974–977. IEEE, 2015.
- Malavika Suresh, Avigyan Sinha, and RP Aneesh. Real-time hand gesture recognition using deep learning. International Journal of Innovations and Implementations in Engineering, 1, 2019.
- Parshwa P Patil, Maithili J Phatak, Suharsh S Kale, Premjeet N Patil, Pranav S Harole, and CS Shinde. Hand gesture recognition for deaf and dumb. Hand, 6(11), 2019
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9). (pp. 1-9). IEEE.
- Omkar Vedak, Prasad Zavre, Abhijeet Todkar, and Manoj Patil. Sign language interpreter using image processing and machine learning. International Research Journal of Engineering and Technology (IRJET), 2019.
- R. G. Rajan and M. Judith Leo, "American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning Features," 2020 International Conference on Inventive Computation Technologies (ICICT), 2020, pp. 430-434, doi: 10.1109/ICICT48043.2020.9112481.
- J. Huang, W. Zhou, H. Li and W. Li, "Attention-Based 3D-CNNs for Large-Vocabulary Sign Language Recognition," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 29, no. 9, pp. 2822-2832, Sept. 2019, doi: 10.1109/TCSVT.2018.2870740.
- A. S. Nikam and A. G. Ambekar, "Sign language recognition using image-based hand gesture recognition techniques," 2016 Online International Conference on Green Engineering and Technologies (IC-GET), 2016, pp. 1-5, doi: 10.1109/GET.2016.7916786.
- U. Patel and A. G. Ambekar, "Moment Based Sign Language Recognition for Indian Languages," 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), 2017, pp. 1-6, doi: 10.1109/ICCUBEA.2017.8463901.
- Y. Liao, P. Xiong, W. Min, W. Min and J. Lu, "Dynamic Sign Language Recognition Based on Video Sequence With BLSTM-3D Residual Networks," in IEEE Access, vol. 7, pp. 38044-38054, 2019, doi: 10.1109/ACCESS.2019.2904749.
- https://machinelearningmastery.com/what-is-deep-learning/
- https://www.tensorflow.org/
- https://www.researchgate.net/publication/331421347

ACKNOWLEDGEMENT

It gives me immense pleasure in expressing my heartfelt thanks to the people who were part of this project in numerous ways. I owe my thanks to all those who gave endless support right from the conception of the project idea to its implementation, it would not have materialized without the help of many.

The dedication, hard work, patience and correct guidance makes any task proficient & a successful achievement. Intellectual and timely guidance not only helps in trying productive but also transforms the whole process of learning and implementing into an enjoyable experience.

I would like to thank our Principal "**Dr. Rohini Kelkar**" and vice principal **"Mr. Asif Rampurawala"** for providing this opportunity, a special thanks to our MSc IT coordinator "**Ms. Beena Kapadia**" for their support, blessings and for being a constant source of inspiration to us. With immense gratitude, I would like to convey my special honour and respect to "**Ms. Pallavi Tawade**" **(Project Guide)** who took keen interest in checking the minute details of the project work and guided us throughout the same.

A sincere thanks to the non-teaching staff for providing us with the long lab timings that we could receive along with the books and with all the information we needed for this project, without which the successful completion of this project would not have been possible.

Finally, I wish to avail this opportunity & express a sense of gratitude and love to my friends and my beloved parents for their support, strength and help for everything.

Shlok Shivkar **Vaishnavi Pangam**

DECLARATION

I hereby declare that the project entitled, **"Sign Language Recognition Using Python And OpenCV"** done at Vidyalankar School of Information Technology, has not been in any case duplicated to submit to any other universities for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of

MASTER OF SCIENCE (INFORMATION TECHNOLOGY) to be submitted as final semester project as part of our curriculum.

Shlok Gurudas Shivkar

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

Sign Language is a form of communication used primarily by people hard of hearing or deaf. This type of gesture-based language allows people to convey ideas and thoughts easily overcoming the barriers caused by difficulties from hearing issues.

There have been several advancements in technology and a lot of research has been done to help the people who are deaf and dumb. Aiding the cause, Deep learning, and computer vision can be used too to make an impact on this cause, moreover, this can be extended to creating automatic editors, where the person can easily write by just their hand gestures.

1.2 Problem Statement

Sign language uses lots of gestures so that it looks like movement language which consists of a series of hands and arms motions. For different countries, there are different sign languages and hand gestures. Also, it is noted that some unknown words are translated by simply showing gestures for each alphabet in the word. In addition, sign language also includes specific gestures to each alphabet in the English dictionary and for each number between 0 and 9.

Based on these sign languages are made up of two groups, namely static gesture, and dynamic gesture. The static gesture is used for alphabet and number representation, whereas the dynamic gesture is used for specific concepts. Dynamic also includes words, sentences, etc. The static gesture consists of hand gestures, whereas the latter includes motion of hands, head, or both. Sign language is a visual language and consists of 3 major components, such as finger-spelling, word-level sign vocabulary, and non-manual features. Finger-spelling is used to spell words letter by letter and convey the message whereas the latter is keyword-based. But the design of a sign language translator is quite challenging despite many research efforts during the last few decades.

Also, even the same signs have significantly different appearances for different signers and different viewpoints. This work focuses on the creation of a static sign language translator by using an object detection algorithm and producing an audio output to the user which allow differently abled people to communicate easily.

1.3 Purpose, Scope & Applicability

1.3.1 Purpose:

The purpose of this project is to facilitate the interaction between deaf & dumb people and normal people to makes the communication between them easier, by translating the sign language to voice or text with high accuracy.
Normal people face difficulty in understanding their language. Hence there is a need for a system which can deal with this situation. The system recognizes the different signs, gestures and conveys the information to the normal people. It bridges the gap between physically challenged people and normal people.

1.3.2 Scope:

The future of object detection technology is in the process of proving itself, and much like the original Industrial Revolution, it has the potential, at the very least, to free people from tedious jobs that will be done more efficiently and effectively by machines. It will also open up new avenues of research and operations that will reap additional benefits in the future. Thus, these challenges circumvent the need to a lot of training requiring a massive number of datasets to serve more nuanced tasks, with its continued evolution, along with the devices and techniques that make it possible, it could soon become the next big thing in the future.

1.3.2 Applicability:

Object detection is a versatile and widely applied technique in computer vision that encompasses various specific applications. These applications range from pedestrian detection, animal detection, and vehicle detection to people counting, face detection, text detection, pose detection, and number-plate recognition etc.

Pedestrian detection focuses on detecting human figures in images or videos, with applications in surveillance systems, driver assistance systems, and crowd analysis.

Animal detection extends the concept of object detection to identify and locate specific animal species in visual data.

Text detection focuses on detecting and extracting text regions from images or scenes. This application finds utility in optical character recognition (OCR), document analysis, image-to-text conversion, and text-based information retrieval.

In summary, object detection encompasses a wide range of specific applications, including pedestrian detection, animal detection, vehicle detection, and people counting, face detection, text detection, pose detection, and number-plate recognition. Each of these applications addresses unique challenges and serves diverse purposes in various domains, contributing to improved safety, efficiency, and automation in different fields.

1.4 Feasibility Study

A significant consequence of starter examination is the affirmation that the framework request is feasible. This is possible just if it is viable inside limited resource and time. The various potential outcomes that must be dismembered are-

- Operational Feasibility.
- Economic Feasibility.
- Technical Feasibility

1.4.1 Economic feasibility :-

Monetary Feasibility or Cost-advantage is an assessment of the budgetary resistance for a PC based endeavour. As gear was presented from the most punctual beginning stage and for heaps of purposes along these lines the cost on the undertaking of hardware is low.

1.4.2 Technical feasibility :-

As demonstrated by Roger S. Pressman, Technical Feasibility is the assessment of the specific resources of the affiliation. The system is made for the stage independent condition. Python code, Html, CSS, coming up short immediately of visual studio code are used to develop the structure. The structure is really down to earth for development and can be made with the present office.

1.4.3 Operational feasibility :-

Operational Feasibility deals with the examination of prospects of the framework to be made. This framework operationally assists customers in sufficiently foreseeing stock estimation of an association, with the objective that customers can settle on up their stock exchanging decisions similarly as improve the gauge model reliant on the comprehension.

1.5 Objectives

Object tracking is considered as an important task within the field of Computer Vision. The invention of faster computers, availability of inexpensive and good quality video cameras and demands of automated video analysis has given popularity to object tracking techniques. This project aims at detecting hand gestures which will help deaf people to communicate through gestures and gives accuracy of that gesture along with an audio output. Investigate how the sign language recognition operates. To analyse the current methods used in sign language recognition. To develop a model for sign language recognition to reduce the communication gap between normal and deaf people.

CHAPTER 2: SURVEY OF TECHNOLOGIES

2.1 Introduction.

The domain analysis that we have done for the project mainly involved understanding the neural networks. These methods for making sign recognition takes help from various domain experts and brings an easier approach for differently abled people to communicate with the rest of us. In this project, we are capturing hand gestures through a webcam and converting this image into a text and audio based output which to has the potential grow beyond just basic communication.
With this research, we hope we can build better communication with deaf people. Because of the complexity and diversity of Sign Language. It makes this topic even harder. Sometimes Sign Language has some similar sign with different meaning. Which is, it made Deep Learning harder to recognize them. And then we also should consider differences between people who do Sign Language.

2.2 Literature Review.

Over the past two decades many important changes have taken place in the environment of Deep Learning and Object Detection Methodologies.

We have discussed about some of the methods for implementing a sign language to text/voice conversion system without using handheld gloves and sensors, by capturing the gesture continuously and converting them to voice.

Turkish Journal of Computer and Mathematics Education Research Article

In this method, only a few images were captured for recognition. The design of a communication aid for the physically challenged. Design of a communication aid for physically challenged. The system was developed under the MATLAB environment. It consists of mainly two phases via training phase and the testing phase. In the training phase, the author used feed forward neural networks.
The problem here is MATLAB is not that efficient and also integrating the concurrent attributes and along with these a few more prominent papers have been researched.

Dynamic Sign Language Recognition Based on Video Sequence with BLSTM-3D Residual Networks Dynamic sign language recognition

It is a method based on analysing the features of hand shapes and motion trajectory with each hand gesture is a normal solution to the problems of dynamic sign language recognition. Increased researchers have focused on sign language recognition based on video sequence. For the method of dynamic sign language recognition based on video sequence, sensors are not necessary elements.

Moment Based Sign Language Recognition for Indian Languages Image Capture

This is the first step in sign recognition Web camera is used to capture the hand gesture Image Preprocessing Image preprocessing contains cropping, filtering, brightness & contrast adjustment & many more. To do such process Image enhancement, Image cropping & Image segmentation methods are used. Feature Extraction Feature extraction is a very useful step to create the database of sign recognition

American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning

Features In this paper, a skin colour-based segmentation method is used to segment the images and these resultant segmented images are supplied to the proposed deep learning model as shown. In this method a deep learning model called VGG-19 is used to extract the features. Afterward, the local features taken from the hand-crafted methods and deep features from deep neural net concatenated serially, finally a support vector machine (SVM) is used to classify the signs.

Sign Language Interpreter using Machine Learning and Image Processing

Pham Microsoft Kinect is used by the Hai to interpret Vietnamese Sign Language. The user must align himself with Kinect's field of view and then conduct sign language movements in the suggested system. Using multiclass Support Vector Machine, it can distinguish both dynamic and static gestures. The gesture features are retrieved, filtered out and normalize on Euclidean distance during recognition.

2.3 Comparative Analysis.

Computer recognition of sign language is an important research problem for enabling communication with hearing impaired people. This project introduces an efficient and fast algorithm for identification of the number of fingers opened in a gesture representing an alphabet of the Binary Sign Language.

The system does not require the hand to be perfectly aligned to the camera. The project uses image processing system to identify, especially English alphabetic sign language used by the deaf people to communicate. The basic objective of this project is to develop a computer based intelligent system that will enable dumb people significantly to communicate with all other people using their natural hand gestures.

The idea consisted of designing and building up an intelligent system using image processing, machine learning and artificial intelligence concepts to take visual inputs of sign language's hand gestures and generate easily recognizable form of outputs. Hence the objective of this project is to develop an intelligent system which can act as a translator between the sign language and the spoken language dynamically and can make the communication between people with hearing impairment and normal people both effective and efficient. The system is we are implementing for Binary sign language but it can detect any sign language with prior image processing

2.4 Research Gap.

As far as our project goes, comparing it with various papers and studies done online people and domain experts have been working on these kinds of methods with different applications have covered the various technological gap but the idea that we are implementing these techniques on generally shows a great research gap as there have been very few instances of using the methods of obtaining an audio output or a verbal message. During past years many researchers have given Sign language recognition is challenged by problems, such as accurate tracking of hand gestures, occlusion of hands, and high computational cost.

CHAPTER 3: REQUIREMENTS AND ANALYSIS

3.1 Problem Definition.

Sign language, as a form of communication, encompasses a rich array of gestures that convey meaning through intricate hand and arm movements. It serves as a distinct language system, with various countries having their own unique sign languages and hand gestures. Interestingly, sign language often involves the translation of unknown words by visually representing each alphabet in the word through gestures. Moreover, specific gestures are assigned to each alphabet in the English dictionary, as well as to numbers ranging from 0 to 9.

Developing a sign language recognition model can pose several challenges due to the unique characteristics and complexities of sign language.

Obtaining a diverse and comprehensive dataset of sign language gestures can be challenging. Sign language includes a vast vocabulary and regional variations, making it essential to have a representative dataset that covers different signs, hand shapes, movements, and expressions. Collecting and annotating such a dataset can be time-consuming and requires expertise in sign language.

Sign language is a visual language that relies on the movement and sequencing of gestures. Capturing the temporal dynamics accurately is crucial for recognizing and interpreting signs. Models need to capture and represent the temporal aspects of sign language effectively, considering the timing, duration, and smoothness of hand movements.

Sign languages can be broadly classified into two groups: static gestures and dynamic gestures. Static gestures are predominantly used for alphabet and number representation, whereas dynamic gestures are employed to express specific concepts. Dynamic gestures involve not only hand motions but also movements of the head or both. Being a visual language, sign language encompasses three primary components: finger-spelling, word-level sign vocabulary, and non-manual features.

Finger-spelling allows individuals to spell out words letter by letter, effectively conveying their intended message. On the other hand, word-level sign vocabulary relies on a keyword-based approach, where signs represent entire words or phrases. Additionally, non-manual features such as facial expressions, body language, and gaze direction play a significant role in sign language, enhancing the conveyed meaning and adding nuance to the communication.

Overcoming these challenges is crucial for creating effective sign language translation systems that can bridge the communication gap between individuals who use sign language and those who do not.

3.2 Requirements Specification / Analysis.

3.2.1 Functional Requirements

- Our model must work and produce the results in real time.
- It should maintain a fairly consistent and concise detection rate.
- The systems speaker should have crisp audio quality.
- Video quality of the camera must be good.
- The overall system should have enough memory so as to not cause any lag when the program is running

3.2.1 Non-Functional Requirements

- **Usability :** The system must be easy to use and the UI design must be fairly simple.
- **Reliability :** Our models data should be accurate and reliable so as to not cause any miscommunication.
- **Scalability :** The model is highly scalable and new instances of objects can be added or removed and the model can be trained as per our needs which adds a lot of room for future upgrades.
- **Persistent Storage:** Adequate storage must be assigned to the program to avoid any scaling problems or avoid lag in some cases.

3.3 Planning and Scheduling.

Planning

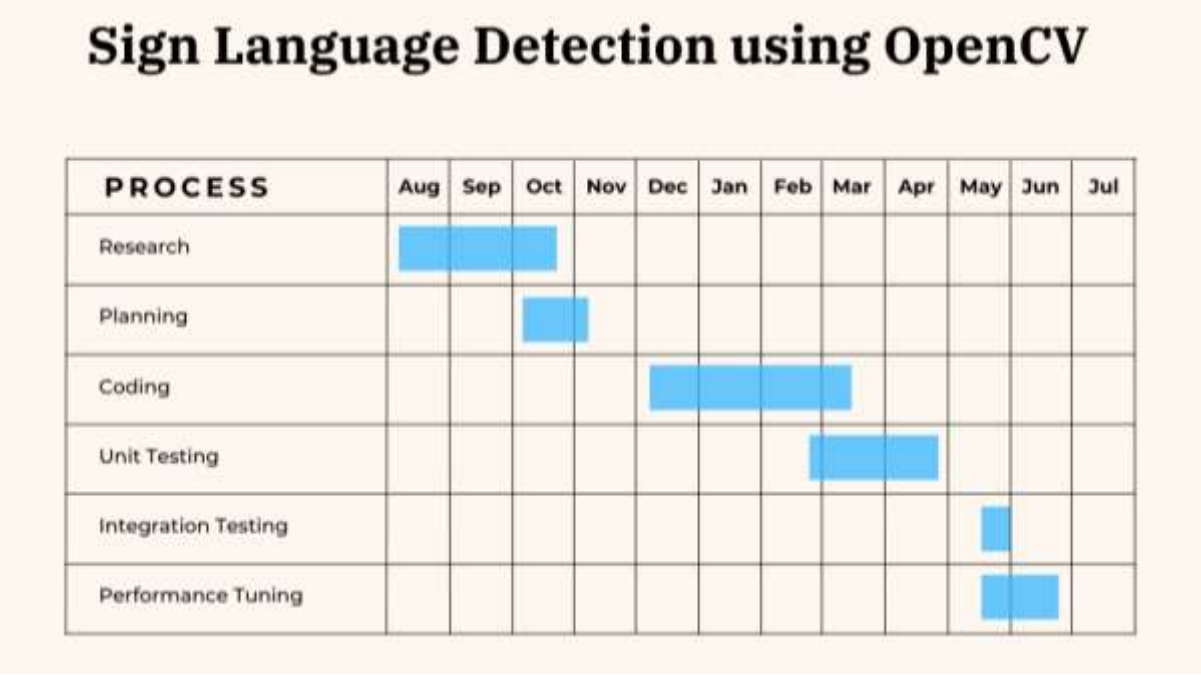
Our objective is to develop a robust and versatile object detection model specifically designed to detect hand signs. This model will not only identify the hand signs accurately but also display the recognized sign on the screen, accompanied by its corresponding confidence or accuracy level. Our overarching aim is to create a highly adaptable and portable model that can seamlessly operate across various platforms and cloud environments, ensuring its universal usability.

To accomplish this, we have devised a comprehensive plan that involves leveraging the power and scalability of cloud computing for training our model. By harnessing the computational resources available in the cloud, we can efficiently train our model on extensive datasets, encompassing a wide range of hand signs, diverse lighting conditions, and varying backgrounds. This extensive training regimen will enable our model to capture the intricacies and complexities inherent in sign language, ensuring high accuracy and robust performance.

Scheduling

Below Gantt chart shows us the schedule of the project. It is a diagram representing the time taken for each task and can be easily prepared using Microsoft Word and Excel as well as some online applications. These types of charts help in the scheduling of the project after proper planning of the project is done. This scheduling helps to find out the estimated and the actual time period of each stage of the project. Here we are going to show 1 Gantt chart: showing the schedule of the documentation

Gantt Chart:



3.4 Software and Hardware Requirements.

3.4.1 Software Requirements:

- Python (3.10.3 and above)
- IDE : Jupyter Notebook / Visual Studio Code / Google Colab
- Numpy (version 1.16.5+)
- cv2 (openCV) (version 3.4.2+)
- TensorFlow (As Keras uses TensorFlow in backend and for image preprocessing) (version 2.0.0)
- Various dependencies (most of the dependencies would be installed automatically but some of them need to be installed manually, you can check the specific required version of each library mentioned below)

Packages	Version
absl-py	1.4.0
apache-beam	2.48.0
asttokens	2.2.1
astunparse	1.6.3
avro-python3	1.10.2
backcall	0.2.0
cached-property	1.5.2
cachetools	5.3.0
certifi	2023.5.7
charset-normalizer	3.1.0
chex	0.1.7
colorama	0.4.6
comm	0.1.3
contextlib2	21.6.0
contourpy	1.0.7
cycler	0.11.0
Cython	3.0.0b3
debugpy	1.6.7
decorator	5.1.1
dm-tree	0.1.8
etils	1.3.0
executing	1.2.0
flatbuffers	23.5.9
flax	0.6.11
fonttools	4.39.4
gast	0.4.0
gin	0.1.6
gin-config	0.1.1
google-auth	2.18.0
google-auth-oauthlib	0.4.6
google-pasta	0.2.0
grpcio	1.54.2
h5py	3.8.0
idna	3.4
importlib-resources	5.12.0
ipykernel	6.23.0

ipython	8.13.2
jax	0.4.13
jaxlib	0.4.13
jedi	0.18.2
jupyter_client	8.2.0
jupyter_core	5.3.0
keras	2.11.0
kiwisolver	1.4.4
libclang	16.0.0
lvis	0.5.3
lxml	4.9.2
Markdown	3.4.3
markdown-it-py	3.0.0
MarkupSafe	2.1.2
matplotlib	3.7.1
matplotlib-inline	0.1.6
mdurl	0.1.2
ml-dtypes	0.1.0
msgpack	1.0.5
nest-asyncio	1.5.6
numpy	1.23.5
oauthlib	3.2.2
object-detection	0.1
opencv-python	4.7.0.72
opt-einsum	3.3.0
optax	0.1.5
orbax-checkpoint	0.2.6
packaging	20.9
pandas	2.0.1
parso	0.8.3
pickleshare	0.7.5
Pillow	9.5.0
pip	23.1.2
platformdirs	3.5.1
portalocker	2.7.0
prompt-toolkit	3.0.38
protobuf	3.19.6
psutil	5.9.5
pure-eval	0.2.2
pyasn1	0.5.0
pyasn1-modules	0.3.0
pycocotools	2.0.6
Pygments	2.15.1
pyparsing	2.4.7
PyQt5	5.15.9
PyQt5-Qt5	5.15.2
PyQt5-sip	12.12.1
python-dateutil	2.8.2
pytz	2023.3
pywin32	306
PyYAML	6.0
pyzmq	25.0.2
regex	2023.6.3
requests	2.30.0
requests-oauthlib	1.3.1
rich	13.4.2
rsa	4.9
sacrebleu	2.2.0

scipy	1.10.1
setuptools	63.2.0
six	1.16.0
slim	0.1
stack-data	0.6.2
tabulate	0.9.0
tensorboard	2.11.2
tensorboard-data-server	0.6.1
tensorboard-plugin-wit	1.8.1
tensorflow	2.11.1
tensorflow-addons	0.20.0
tensorflow-estimator	2.11.0
tensorflow-hub	0.12.0
tensorflow-intel	2.11.1
tensorflow-io	0.31.0
tensorflow-io-gcs-filesystem	0.31.0
tensorflowjs	3.21.0
tensorstore	0.1.39
termcolor	2.3.0
tf-models-official	2.12.0
tf-slim	1.1.0
toolz	0.12.0
tornado	6.3.1
traitlets	5.9.0
typeguard	2.13.3
typing_extensions	4.5.0
urllib3	1.26.15
wcwidth	0.2.6
Werkzeug	2.3.4
wget	3.2
wheel	0.40.0
wrapt	1.14.1

3.4.2 Hardware Requirements:

- **Processor (CPU):** A powerful multi-core processor is essential for efficient object detection. A minimum of a quad-core CPU is recommended, but a higher core count, such as an octa-core or higher, will yield better performance. Intel Core i3 and above is recommended to run this project smoothly.

- **Graphics Processing Unit (GPU) *optional*:** Object detection models can significantly benefit from GPU acceleration. A dedicated GPU with CUDA support is highly recommended, but is purely optional. The specific GPU requirements depend on the model architecture and dataset size. For example, models like YOLO or SSD can benefit from high-end GPUs such as NVIDIA GeForce GTX series or NVIDIA GeForce RTX series.

- **Webcam:** The resolution of the webcam determines the quality of the captured video feed. Higher resolutions can provide more detailed images but may require more processing power. A minimum resolution of 720p (1280x720 pixels) is recommended for most object detection tasks, but higher resolutions like 1080p (1920x1080 pixels) or even 4K (3840x2160 pixels) can provide better image quality if available.
    - **Frame Rate:** The frame rate refers to the number of frames per second (fps) that the webcam can capture. A higher frame rate allows for smoother video playback and more accurate object detection. A minimum frame rate of 30 fps is typically sufficient for most applications, but higher frame rates like 60 fps can be advantageous in scenarios where fast-moving objects need to be detected accurately.

- **Memory:** Sufficient memory is crucial for storing the model, input data, and intermediate computations. The minimum recommended RAM size is 8 GB, but for larger models or datasets, 16 GB or more may be necessary..

- **Storage:** Adequate storage space is required to store the model, input data, and any intermediate results. A solid-state drive (SSD) is recommended for faster data access and model loading times.

CHAPTER 4: SYSTEM DESIGN

4.1 Basic Modules

- **Image collection for object detection:** This involves gathering a dataset of images that contain the objects you want to detect, in this case, sign language gestures. The images should represent a variety of signs performed by different individuals, capturing different angles, lighting conditions, and backgrounds. The purpose is to provide diverse and representative data for training an object detection model. The signs that we are including in this project demonstration are: Thumbs Up, Thumbs Down, Hello, Thank You, Call, What, Your, Name, S or Yes, H, L, O, K or Peace.
  The hand sign for S and Yes is the same hence it has been merged together to form a single label called S_or_Yes, same is the case for K and Peace which is merged to form the label K_or_Peace

- **Labelling the captured images:** Once the images are collected, the next step is to label them. Image labelling involves annotating or marking the regions of interest within the image that correspond to the sign language gestures. Each gesture should be delineated with a bounding box or polygon to indicate its location within the image. This step is crucial for training an object detection model because it helps establish a ground truth for the training data.

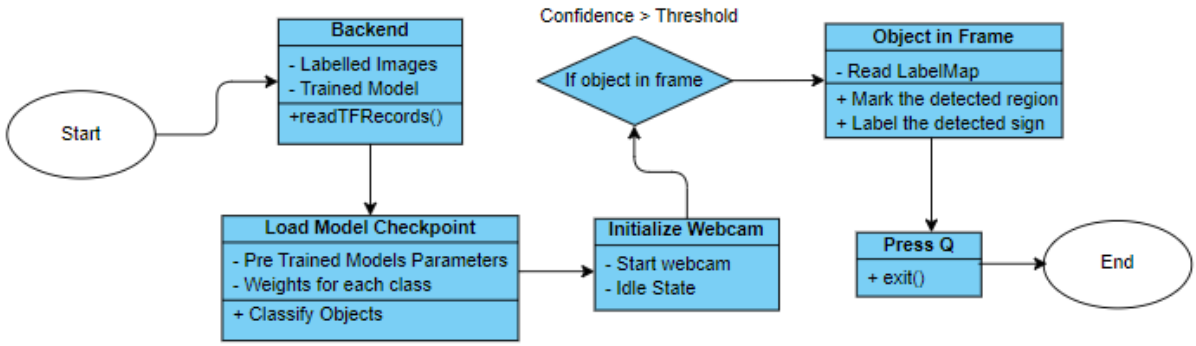
- **Training the model:** After the images are collected and labelled, the training phase begins. In this module, the labelled images are used to train an object detection model. The model learns to recognize and classify the sign language gestures by analysing the visual features present in the labelled images. Training typically involves feeding the labelled images into a deep learning framework, such as TensorFlow or PyTorch, and using a specific object detection architecture, such as YOLO, SSD, or Faster R-CNN, to learn the patterns and characteristics of the gestures.
  In this project we are using TensorFlow and SSD (Single-Shot Detector) as the Deep Learning Framework and the Object Detection Model respectively.

- **Real Time Detection:** Once the model is trained, it can be used for real-time sign language detection. This module involves using the trained model to process live video or webcam feeds, analysing each frame in real-time to detect and recognize sign language gestures

- **Image Detection:** Once the model is trained, we can use it for detecting objects or hand signs in our case from individual images.

4.2 Diagrams

4.2.1 Class Diagram :-

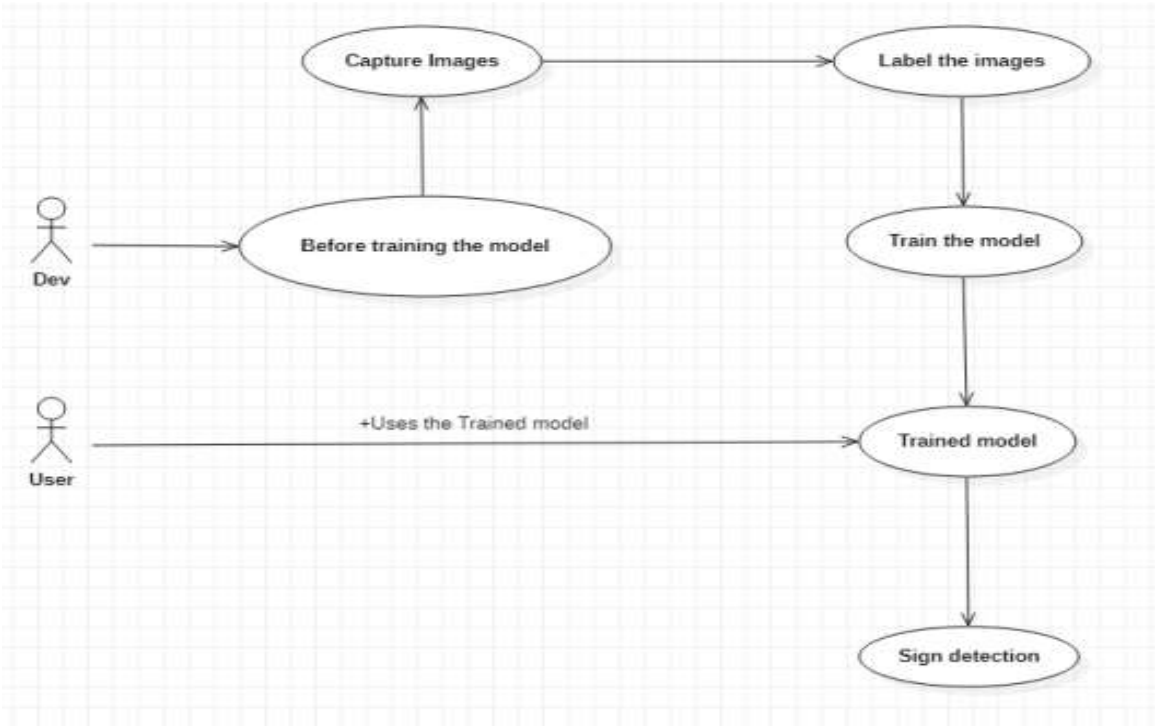


Description of Class Diagram:

Upon successful training of the object detection model, it can be utilized for real-time detection. The system leverages TFRecords, which were created during the training phase, to provide input data for inference. Subsequently, the model checkpoint file, encompassing the parameters and weights for each detectable class, is loaded.

To initiate real-time detection, the webcam is initialized and placed in an idle state. As objects enter the frame, the system examines whether the detected object surpasses a predefined confidence threshold. In the event that the threshold is exceeded, the object is annotated with the corresponding label and the confidence of the detection. Additionally, the region of the object is visually marked, enhancing the interpretability of the results. By implementing this process, real-time object detection becomes feasible, enabling the system to analyse live video feeds and provide accurate and confidence-based identification of objects within the captured frames.

4.2.2 Use Case Diagram :-

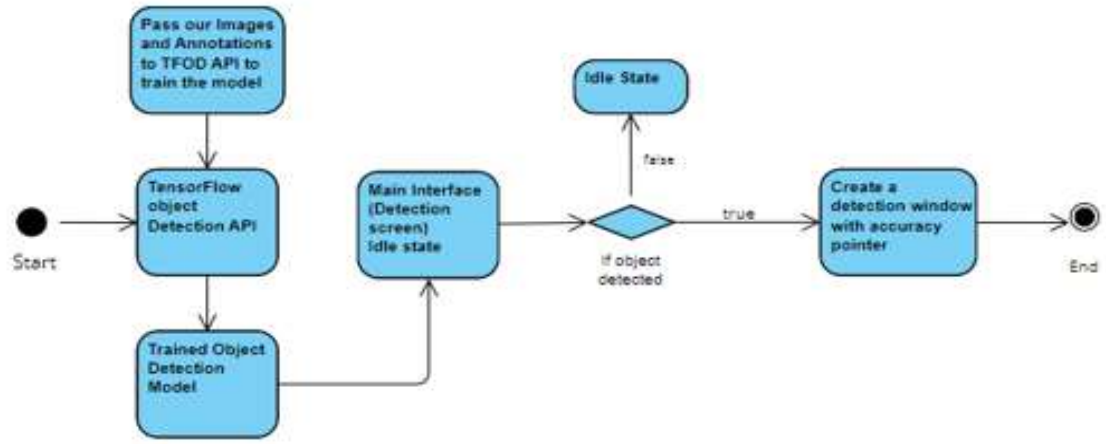


Description of Use Case Diagram:

Use case diagrams referred as a behaviour model or diagram. It simply describes the process and the mode of usage of the application, initially we have to create an object detection system, by collecting images and then labelling them and use it for training our model.

Then the trained model can perform well with high accuracy if we have correctly labelled the images and trained on them for a brief amount of time.

4.2.3 Activity Diagram:



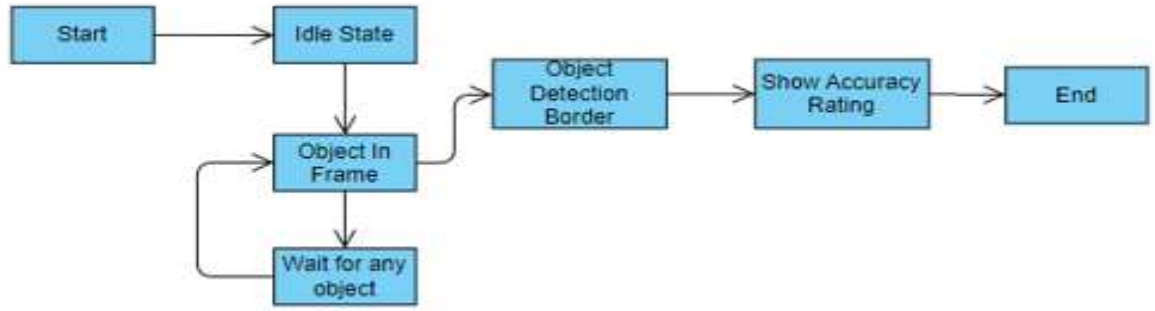
Description of Activity Diagram:

The presented Activity Diagram illustrates the fundamental workflow of the application. Upon launching the application, the camera component is initialized, entering an idle state as it awaits the detection of objects specified in our Label Map.

Once an object of interest is identified, the application promptly responds by visually highlighting the detected object on the screen. This highlighting is accomplished by enclosing the object within a bordered region, referred to as the detection window. Additionally, the application provides an indication of the accuracy of the hand sign recognition by displaying the corresponding confidence level alongside the highlighted object.

By adhering to this logic, the application ensures an efficient workflow, seamlessly leveraging the camera input to detect and visually represent hand signs with accuracy and precision.

4.2.4 Block Diagram :-



Description of Activity Diagram :-

When we start the camera for detection, it is initially in the idle state and waits for any object to be come in frame. As objects enter the frame, the system examines whether the detected object surpasses a predefined confidence threshold. In the event that the threshold is exceeded, the object is annotated with the corresponding label and the confidence of the detection. Additionally, the region of the object is visually marked, enhancing the interpretability of the results.

By implementing this process, real-time object detection becomes feasible, enabling the system to analyse live video feeds and provide accurate and confidence-based identification of objects within the captured frames.

4.3 Test Cases Design.

4.3.1 Step wise test cases

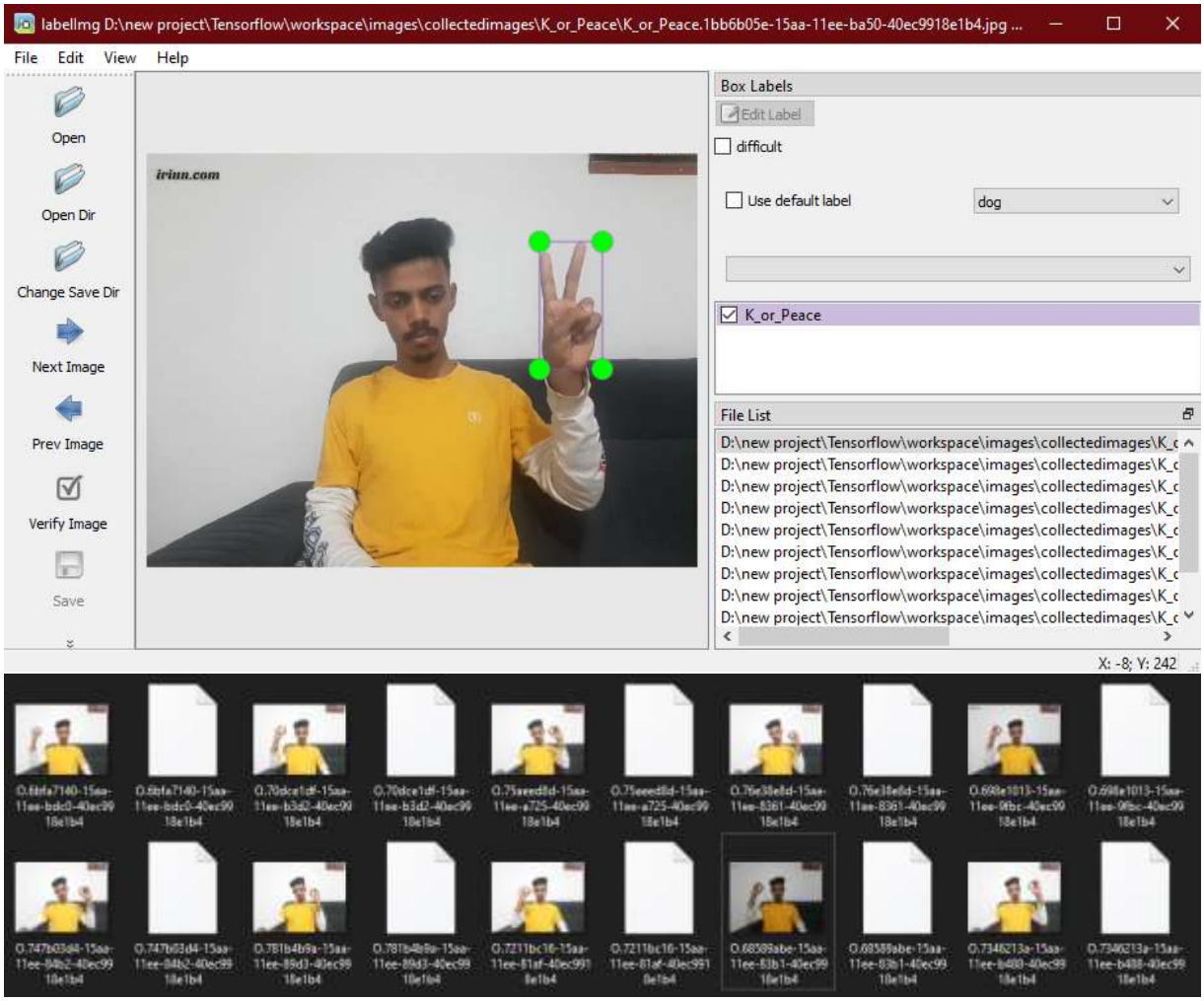
Test Case No.	Description	Expected Result	Pass / Fail	Actual Result
1	Create a virtual environment	Environment must be created and activated	Pass	Environment created successfully. Activation and deactivation successful
2	Verify proper installation of all the dependencies using a verification script	All dependencies must be installed perfectly	Pass	Installation of all dependencies verified successfully
3	Creating instance for camera or webcam	Should be able to initialize camera	Pass	Instance created successfully
4	Capture images	Should capture images at fixed intervals	Pass	Captured images successfully
5	Label Images	Prepare the entire library so that we can start detection	Pass	Labelling done
6	Detection from images	Sign must be detected successfully from images	Pass	Signs from images are successfully detected
7	Detection in real time	Our Program should detect the signs accurately	Pass	All the signs are being detected successfully

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Implementation Approaches

To facilitate object detection on our local machine, we will begin by installing and setting up the necessary components. In this case, we will utilize the TensorFlow Object Detection API, which greatly simplifies the process of building an object detection model. This API provides pre-built functionality for pre-processing, post-processing, visualization utilities, and optimizes the overall object detection workflow.

The next step involves collecting images using a webcam and labelling them. Labelling entails marking the objects of interest, in this case, drawing bounding boxes around the hand signs present in the images. These labelled images serve as the training data for our object detection model.



We label the images by using the labelIMG repository, It is a graphical image annotation tool, written in Python and uses Qt for its graphical interface.

Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet.

Subsequently, we proceed to train the model using the labelled images. We select an appropriate architecture from the TensorFlow Model Zoo, a repository containing pre-trained detection models on the COCO 2017 dataset. Our choice for this project is the SSD (Single-Shot Detector) MobileNet V2 FPNLite 320x320 model, which strikes a balance between speed and precision.

Model name	Speed (ms)	COCO mAP	Outputs
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes

To facilitate the training process, we create a label map, which acts as a reference for all the identified signs or labels, assigning each label a unique ID. Additionally, we generate a TFRecords file, a binary file format that optimizes data storage and retrieval, enhancing training efficiency for our custom object detection model.

The TensorFlow Model Zoo offers a range of models, and the pipeline.config file plays a vital role in defining the architecture and behaviour of our chosen model. This configuration file acts as a baseline, incorporating the feature extractor and performing data augmentation. We customize the pipeline.config file to align with our specific requirements and objectives.

Once all the necessary preparations are complete, we commence the training process. This involves feeding the labelled images and corresponding labels into the model, enabling it to learn and improve its ability to detect hand signs accurately.

Upon completion of the training process, we can utilize the trained model to initiate the detection of hand signs. Leveraging the knowledge acquired during training, the model analyzes input images or video frames and identifies the presence of hand signs, providing real-time detection results.

By following this systematic approach, we establish a robust object detection pipeline, enabling the accurate identification of hand signs using our custom-trained model.

5.2 Coding Details

Within our project, two pivotal code files play a critical role in ensuring its success and functionality. The first code file is responsible for the essential image collection and labelling process, consolidating these tasks into a cohesive workflow. By encapsulating image collection and labelling within a single code file, we streamline the data acquisition process, facilitating subsequent stages of model training and detection.

The image collection aspect of the code file enables the systematic acquisition of images using our webcam, ensuring a diverse and representative dataset for training and evaluation. Simultaneously, the code file incorporates labelling functionality, allowing us to annotate and mark the objects of interest, specifically hand signs in this context. Through this unified approach, the code file empowers us to efficiently gather and label the necessary images, laying a solid foundation for subsequent model development.

The second code file focuses on critical aspects of the project, including verifying dependencies, creating the label map, updating the configuration for transfer learning, training the model, and enabling both image-based and real-time detections. This code file acts as a comprehensive toolkit, encompassing various functionalities essential for the success of our object detection project.

To ensure a smooth and error-free execution, the code file systematically verifies dependencies, ensuring that all required libraries, packages, and modules are present and compatible with our project's specifications. This step mitigates potential compatibility issues and ensures a robust working environment for subsequent tasks.

Creating the label map within the code file involves mapping the detected objects, specifically hand signs, to their corresponding labels. This mapping facilitates the understanding and interpretation of the model's output, enabling us to identify and differentiate between different hand signs during the detection phase.

Another crucial task handled by this code file is the updating of the configuration for transfer learning. Transfer learning leverages pre-trained models as a starting point, significantly accelerating the model development process. By adapting and fine-tuning a pre-existing model architecture, we can efficiently train our model to accurately detect hand signs, benefiting from the pre-learned features and representations captured by the base model.

The code file orchestrates the training process, allowing us to optimize and refine the model's performance through iterative epochs. This entails feeding the labelled dataset into the model, enabling it to learn and adapt its internal parameters to achieve higher accuracy and precision in detecting hand signs.

Furthermore, the code file provides functionalities for detecting hand signs from images, enabling us to evaluate the model's performance on static inputs. Additionally, it facilitates real-time detection, harnessing the power of the model to identify and track hand signs in a dynamic and continuous manner.

By encompassing these crucial functionalities within these two code files, our project benefits from a well-structured and comprehensive framework. This approach empowers us to efficiently handle image collection, labelling, dependency verification, model training, and both image-based and real-time detections, ultimately enabling us to achieve accurate and robust hand sign detection capabilities.

Code :

Image Collection Notebook

1. Import Dependencies

 # Import opencv

```
!pip install opencv-python
import cv2
```

 # Import uuid (In Python, the UUID module provides functionalities to generate universally unique identifiers (UUIDs). UUIDs are 128-bit values that are unique across all devices and time, and they are often used to assign unique identifiers to entities in various systems or applications)

```
import uuid
```

 # Import Operating System

```
import os
```

 # Import time (the time module provides functions that allow you to work with time-related operations. By importing the time module with the import time statement, you gain access to these functions and classes for time-related tasks)

```
import time
```

2. Define Images to Collect

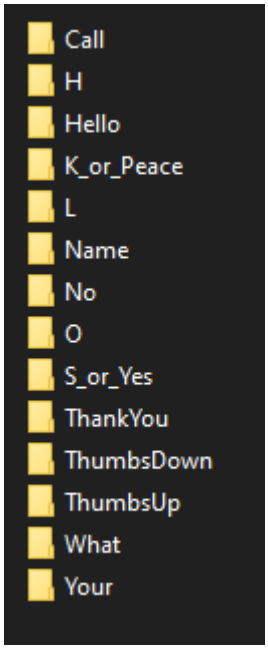
 # Specify the number of labels that you want to capture and the label names

```
labels = ['ThumbsUp', 'ThumbsDown', 'S_or_Yes', 'No', 'ThankYou', 'Hello', 'Call',
'K_or_Peace','Your','Name','What','H','L','O']
number_imgs = 15
```

3. Setup the folders where you want to save the images

 # This is a script that automatically creates the folders of specific labels that are mentioned above

```
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
if not os.path.exists(IMAGES_PATH):
    if os.name == 'posix':
        !mkdir -p {IMAGES_PATH}
    if os.name == 'nt':
         !mkdir {IMAGES_PATH}
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
    if not os.path.exists(path):
        !mkdir {path}
```



4. Capture Images

 # This is a script that keeps on capturing the images using your webcam and are stored in the specified folders mentioned before

```
for label in labels:
    cap = cv2.VideoCapture(1)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_imgs):
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()
        imgname =
os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break
cap.release()
cv2.destroyAllWindows()
```

5. Image Labelling

 # This part of codes states the labelling image path, clones the labelIMG repository onto our local machine and starts it for labelling the images

```
!pip install --upgrade pyqt5 lxml
```

 # Specify path

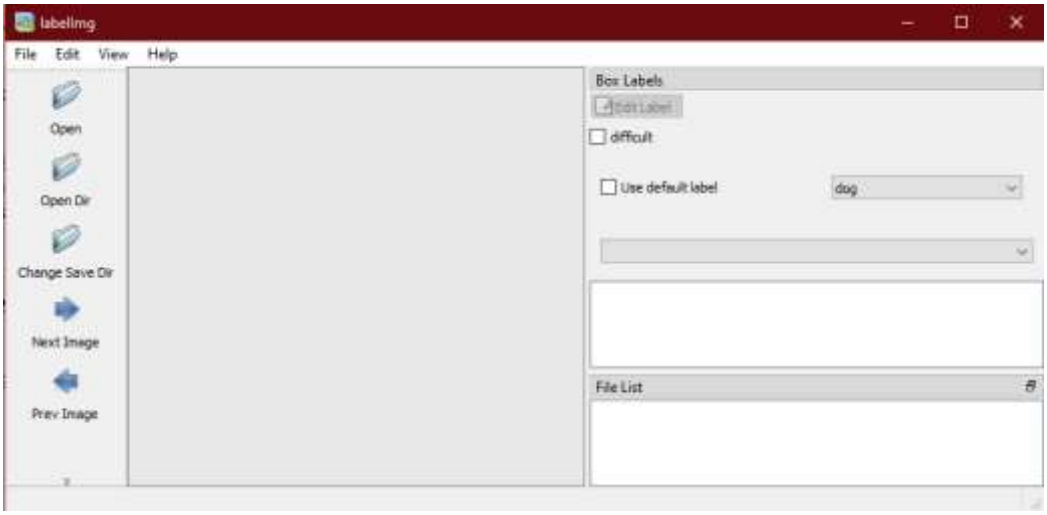
```
LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
```

#clone the repository

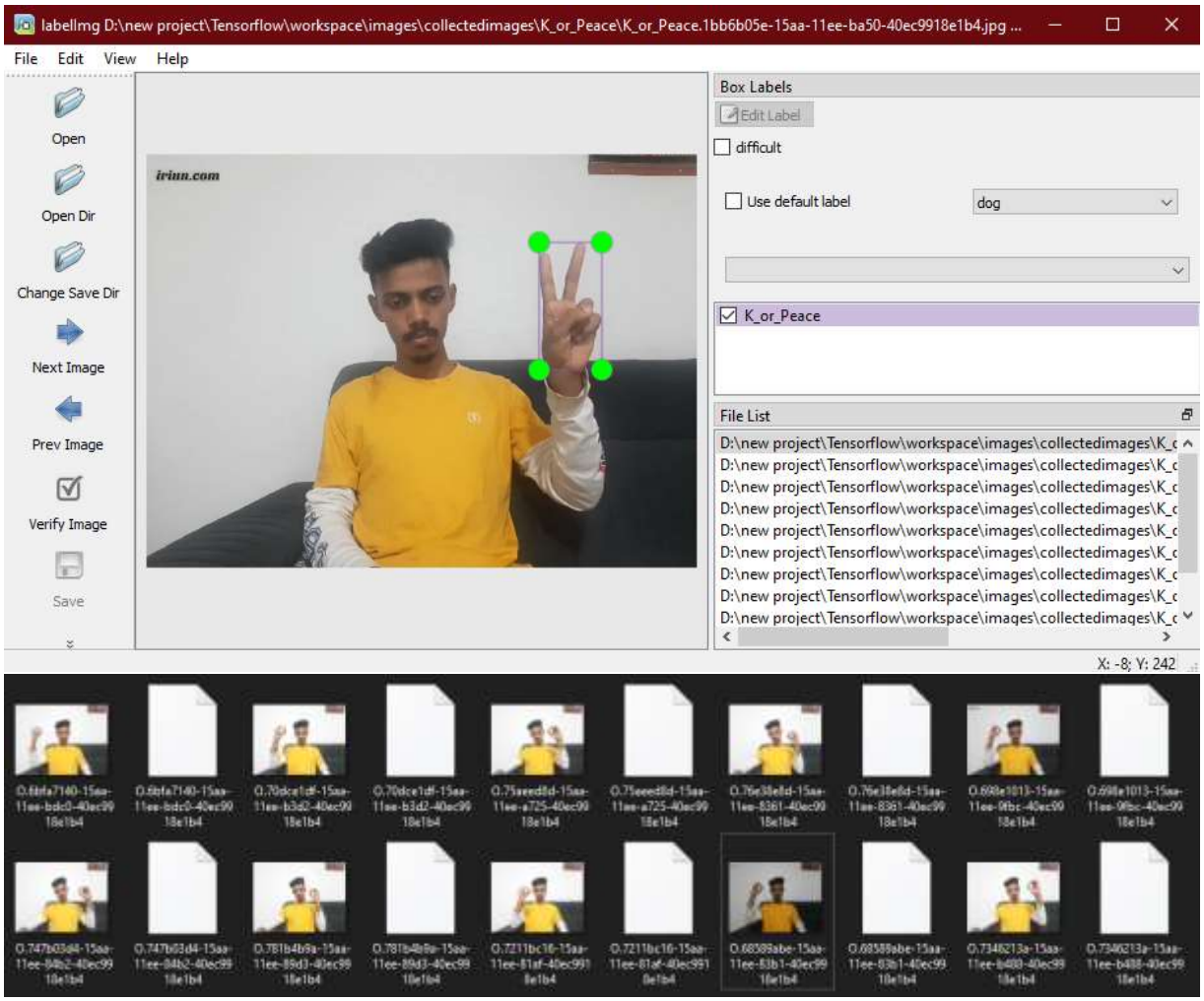
```
if not os.path.exists(LABELIMG_PATH):
   !mkdir {LABELIMG_PATH}
   !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
if os.name == 'posix':
   !make qt5py3
if os.name =='nt':
   !cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc
```

 # Start the software

```
!cd {LABELIMG_PATH} && python labelImg.py
```

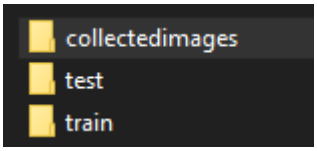


 # Label the images respectively for each label and generate annotations



6. Move the captured images into Training and Testing sets

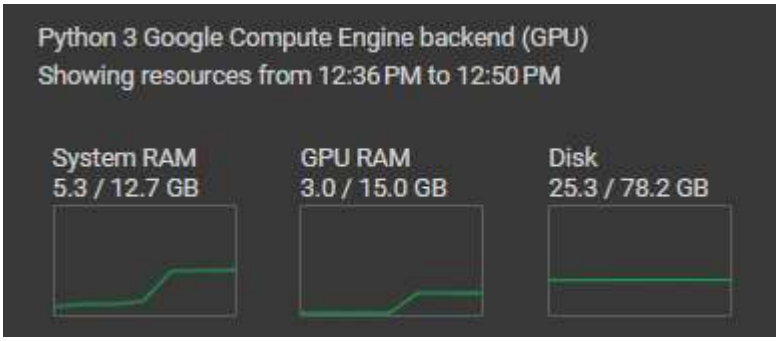
 # Copy the images along with its annotations that are created by labelling them and divide it in the ratio of 80:20 for training set and test set respectively



7. Compress the training set and test set for training it on any cloud platform

 # In our project we will be training the model on Google Colab as it allows us to use its GPU runtime which results in training time of up to 40% to 80% in machines with less powerful hardware

#The steep rise in the graph indicates the kick-off of training phase and states the resources that are being used for training



 # specify the paths to training and test set
TRAIN_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'train')
TEST_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'test')
ARCHIVE_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'archive.tar.gz')

 # Compress the training and test set
!tar -czf {ARCHIVE_PATH} {TRAIN_PATH} {TEST_PATH}

- **Training and detection notebook**

 # Importing Operating System
import os

1. Specify all the paths, files and URL's and create them

#specifying the pre trained model name, tf record name and labelmap name

CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL =
'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'

paths = {
'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
'APIMODEL_PATH': os.path.join('Tensorflow','models'),
'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),

```
    'CHECKPOINT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }

files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models',
CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

 # Create the following paths which are mentioned above

```
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

2. Download TF Models Pretrained Models from TensorFlow Model Zoo and install TensorFlow Object Detection API

 # Import wget. It is a computer tool created by the GNU Project. You can use it to retrieve content and files from various web servers. The name is a combination of World Wide Web and the word get. It supports downloads via FTP, SFTP, HTTP, and HTTPS.

```
import wget
```

#Download the TensorFlow Object Detection API

```
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection')):
        !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

 # Install TensorFlow Object Detection

```
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --
python_out=. && cp object_detection/packages/tf2/setup.py . && python -m pip install .

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-
3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep +
os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --
python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py && python
setup.py build && python setup.py install
    !cd Tensorflow/models/research/slim && pip install -e .
```

 # Run the Verification Script to ensure all the dependencies are installed correctly and are working perfectly as intended

 # Specify the script

```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'builders', 'model_builder_tf2_test.py')
```

 # Running the script and verifying installation

```
!python {VERIFICATION_SCRIPT}
```

 # If you get any errors when running this script then import the appropriate dependencies and run it again until you get an ok message at the end

 # Some of the dependencies that might be required are specified below

```
pip install --upgrade "jax[cuda]" -f https://storage.googleapis.com/jax-
releases/jax_releases.html

pip install "jax[cpu]===0.3.14" -f https://whls.blob.core.windows.net/unstable/index.html
--use-deprecated legacy-resolver

#pip install jax==0.4.13

#pip install -U jax jaxlib
```

```
#!pip install tensorflow –upgrade

#pip install tensorflow

#!pip install tensorflow-addons

#!pip install pyyaml

#pip install matplotlib

#pip install protobuf==3.20.*

#!pip install object-detection

#!pip install gin

#!pip install gin-config==0.1.1

#pip install pytz
```

 # Once we get this message we are ready to proceed with the further steps

```
-------
Ran 24 tests in 35.595s

OK (skipped=1)
```

3. Creating a LabelMap

 # This scripts creates a LabelMap that specifies the labels that are used for detection

```
labels = [{'name':'ThumbsUp', 'id':1},
          {'name':'ThumbsDown', 'id':2},
          {'name':'ThankYou', 'id':3},
          {'name':'S_or_Yes', 'id':4},
          {'name':'No', 'id':5},
          {'name':'K_or_Peace', 'id':6},
          {'name':'Call', 'id':7},
          {'name':'Hello', 'id':8},
          {'name':'H', 'id':9},
          {'name':'L', 'id':10},
          {'name':'O', 'id':11},
          {'name':'Name', 'id':12},
          {'name':'What', 'id':13},
          {'name':'Your', 'id':14}]
```

```
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')
```

 # If you are running on Google Colab you will have to extract the training and test set that we compressed previously

```
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}
```

4. Creating TFRecords File

 # TFRecords is a file format used in TensorFlow for efficient storage and retrieval of large amounts of data during training and inference processes. It is a binary format that allows for optimized reading and writing operations, making it particularly suitable for handling large datasets.

#clone the TFRecord Repository

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord
{paths['SCRIPTS_PATH']}
```

#Generate the TFRecords of Train and Test set

```
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'],
'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'],
'test.record')}
```

5. Copy Models Pipeline.Config to the Training Folder

```
if os.name =='posix':
    !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'],
PRETRAINED_MODEL_NAME, 'pipeline.config')}
{os.path.join(paths['CHECKPOINT_PATH'])}
if os.name == 'nt':
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'],
PRETRAINED_MODEL_NAME, 'pipeline.config')}
{os.path.join(paths['CHECKPOINT_PATH'])}
```

6. Updating the Pipeline.Config that we will use for Transfer Learning

 # Importing the dependencies

```
import tensorflow as tf
import object_detection
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

 # Specifying the pipeline.config path

```
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```

 # Update it according to our correct paths

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)

pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'test.record')]

config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

7. Training the model

 # Generating a training script, this runs the model_main_tf2.py

```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'model_main_tf2.py')
```

 # Specify the training command that starts the training

command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=2000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])

 # When we print this command we get the following code which specifies the locations to our pipeline config, pre trained model and the number of training steps. We can use this directly in the command prompt to train our model as well

python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config --num_train_steps=20000

8. Evaluating the model

 # Generating the command for Evaluation of our model

command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])

 # When we print this command we get the below command

python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config --checkpoint_dir=Tensorflow\workspace\models\my_ssd_mobnet

 # This command again runs the model_main_tf2.py script and takes the newly created model along with the pipeline.config and checkpoints of our model to evaluate it.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.872
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 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 ] = 0.861
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.875
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.875
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.875
 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 ] = 0.844
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.867
INFO:tensorflow:Eval metrics at step 20000
```

```
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.871782
I0628 21:17:49.905550 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP: 0.871782
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
I0628 21:17:49.907548 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
I0628 21:17:49.909543 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): -1.000000
I0628 21:17:49.911564 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): 0.843936
I0628 21:17:49.912534 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (medium): 0.843936
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.861276
I0628 21:17:49.914558 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (large): 0.861276
```

```
INFO:tensorflow:        + DetectionBoxes_Recall/AR@1: 0.875000
I0628 21:17:49.915526 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@1: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@10: 0.875000
I0628 21:17:49.917667 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@10: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100: 0.875000
I0628 21:17:49.918664 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0628 21:17:49.919664 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (medium): 0.843750
I0628 21:17:49.921681 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (medium): 0.843750
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (large): 0.866667
I0628 21:17:50.044589 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (large): 0.866667
```

```
INFO:tensorflow:        + Loss/localization_loss: 0.029558
I0628 21:17:50.046946 17428 model_lib_v2.py:1018]       + Loss/localization_loss: 0.029558
INFO:tensorflow:        + Loss/classification_loss: 0.078346
I0628 21:17:50.047945 17428 model_lib_v2.py:1018]       + Loss/classification_loss: 0.078346
INFO:tensorflow:        + Loss/regularization_loss: 0.067200
I0628 21:17:50.049921 17428 model_lib_v2.py:1018]       + Loss/regularization_loss: 0.067200
INFO:tensorflow:        + Loss/total_loss: 0.175104
```

9. Load Train Model from Checkpoints

 # Importing Dependencies

 # Object Detection is a package build from TensorFlow's object detection API.

```
!pip install object-detection
import object detection

import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
```

 # Load pipeline config and build a detection model

```
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False
)
```

 # Restore the latest checkpoint that we have generated in our model

```
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-21')).expect_partial()
```

```
@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

10. Detect Hand Signs from an Image

 # Import the dependencies, numpy is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

 # Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

#Specifying the category index and image path

```
category_index =
label_map_util.create_category_index_from_labelmap(files['LABELMAP'])

IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'ThumbsDown.0af1bbf9-
1341-11ee-9211-b05cda277e61.jpg')
```

 # Following is a script that allows you to detect hand signs from an image by reading the image path mentioned above

```
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections
```

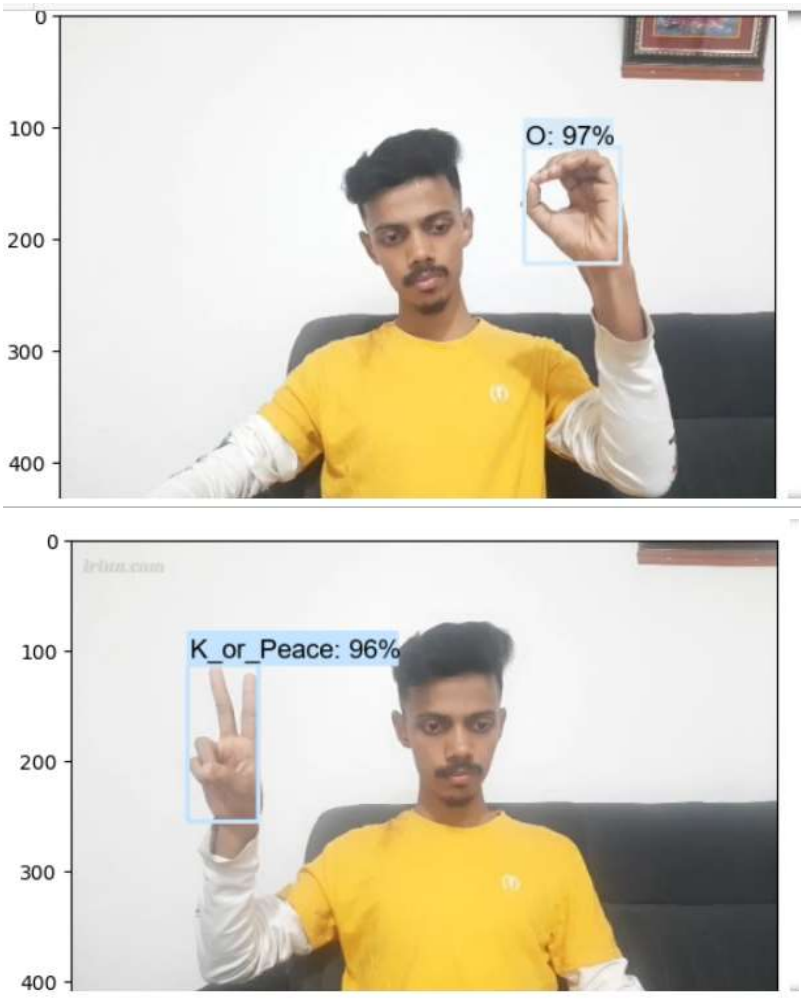
```
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes']+label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.8,
        agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

#This will result in something like this but for the image that you specified before



11. Real Time Detections from the webcam

 # The following script will initialize your webcam and will create a frame that will detect hand signs

#Specify the video capture frame here (0 is generally your webcam, 1 or more is for any external video capture sources that you have attached to your system)

```
cap = cv2.VideoCapture(1)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()
```

```
viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes']+label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.60,
        agnostic_mode=False)

cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

if cv2.waitKey(10) & 0xFF == ord('q'):
    cap.release()
    cv2.destroyAllWindows()
    break
```

5.3 Testing Approaches

Use case diagrams referred as a behaviour model or diagram. It simply describes the process and the mode of usage of the application, initially we have to create an object detection system, by collecting images and then labelling them and use it for training our model.

Then the trained model can perform well with high accuracy if we have correctly labelled the images and trained on them for a brief amount of time.

5.3.1 Unit Testing:

The entire model can be broken down into its individual components that can detect signs from single images. For unit testing we can try out test cases on the individual images and evaluate the results.

Testing Results for each individual hand sign is shown below:

1. Call: Accuracy = 99%

2. Hello: Accuracy = 98%

3. No: Accuracy = 90%

5. Name: Accuracy = 94%

6. Thank You: Accuracy = 99%

7. Thumbs Up: Accuracy = 100%

8. Thumbs Down: Accuracy =99%

9. Your: Accuracy = 90%

10. S_or_Yes: Accuracy = 100%

11. H: Accuracy = 97%

12. L: Accuracy = 92%

13. O: Accuracy = 99%

14. K_or_Peace: Accuracy = 89%

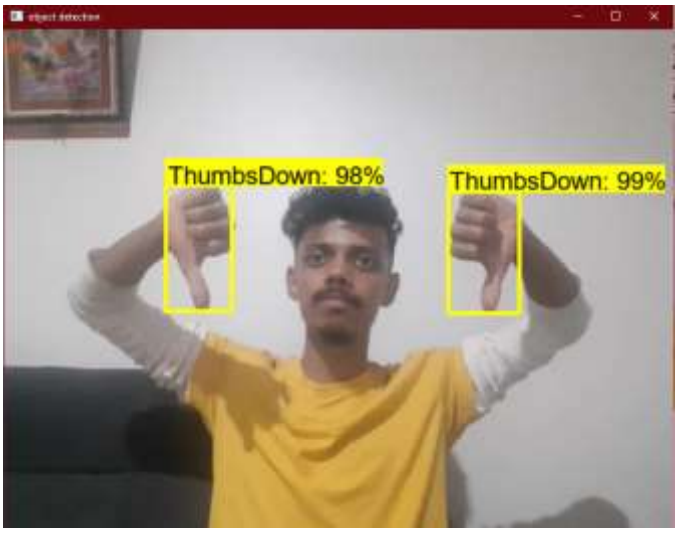
5.3.2 Integration Testing:

The integration testing phase for a sign language recognition model involves the comprehensive assessment of the combined functionality and performance of all the symbols or hand signs in real-time scenarios. This critical phase aims to verify that the model operates seamlessly and effectively as expected when subjected to a variety of sign gestures and their combinations.

1. Hello

2. Thumbs Up

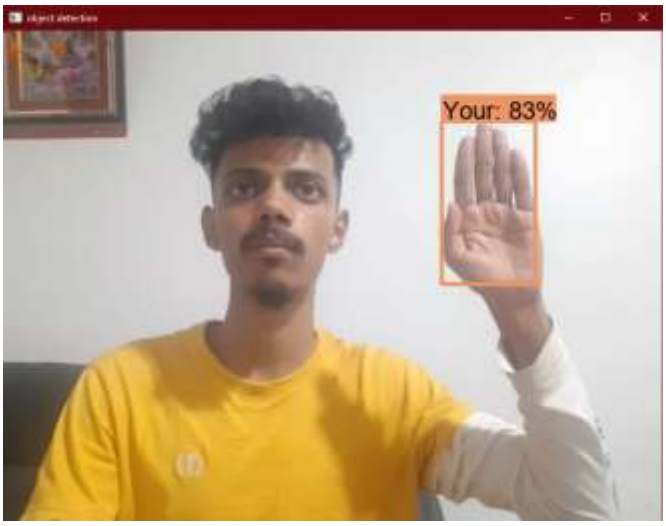
3. Thumbs Down



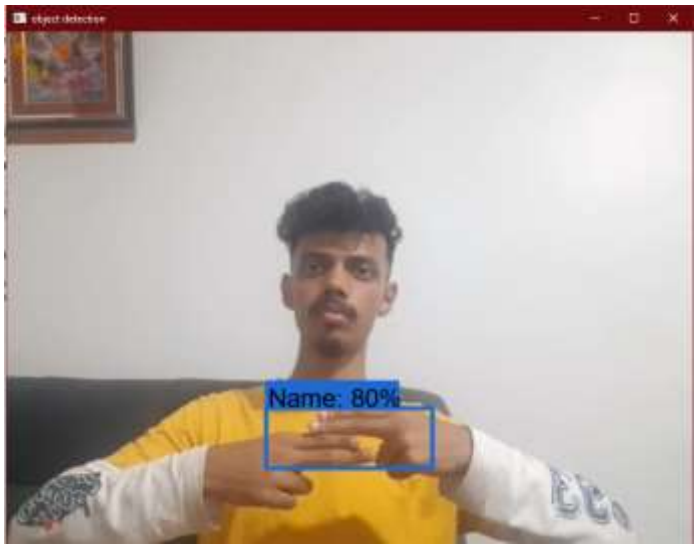
4. No

5. What/s

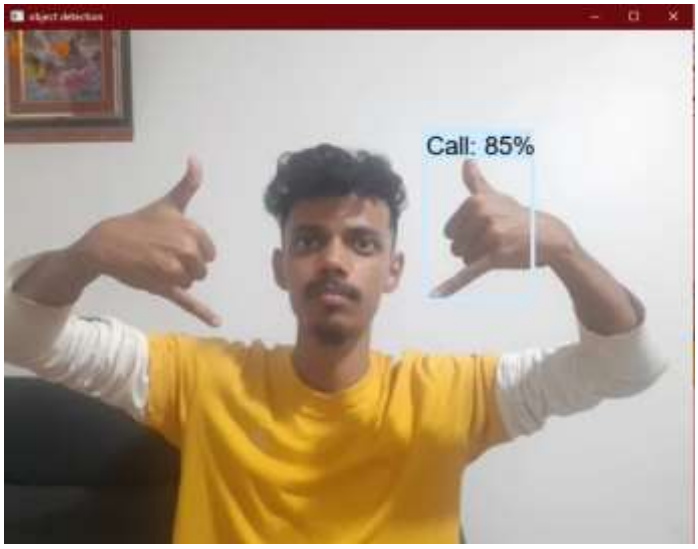
6. Your



7. Name



8. Call



9. ThankYou

10. S_or_Yes

11. H

12. L

13. O

14. K_or_peace

CHAPTER 6: RESULTS AND DISCUSSIONS

6.1 Test Reports

The Performance metrics or performance report of our model can be viewed in the console as well as the TensorBoard

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

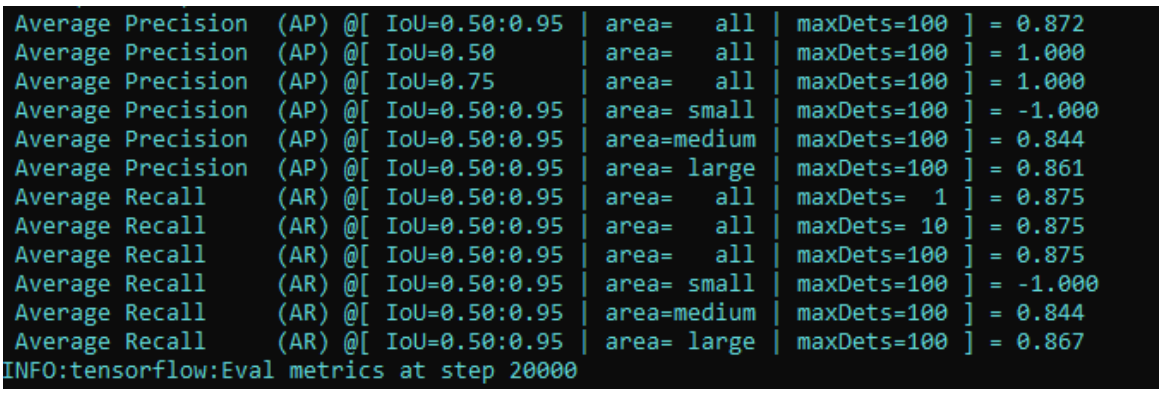
- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs

Performance metrics are shown below:

1. Average Precision and Recall of the model:

Here we can see the average precision and recall varying over different IoU's.

IoU stands for Intersection over Union, which is an evaluation metric commonly used in object detection and image segmentation tasks. It measures the overlap between the predicted region and the ground truth region by calculating the ratio of the intersection area to the union area of the two regions.



2. Mean Average Precision over small medium and large objects:

DetectionBoxes_Precision/mAP (small): mean average precision for small objects (area < 32^2 pixels).

DetectionBoxes_Precision/mAP (medium): mean average precision for medium sized objects (32^2 pixels < area < 96^2 pixels).

DetectionBoxes_Precision/mAP (large): mean average precision for large objects (96^2 pixels < area < 10000^2 pixels).

```
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.871782
I0628 21:17:49.905550 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP: 0.871782
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
I0628 21:17:49.907548 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
I0628 21:17:49.909543 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): -1.000000
I0628 21:17:49.911564 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): 0.843936
I0628 21:17:49.912534 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (medium): 0.843936
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.861276
I0628 21:17:49.914558 17428 model_lib_v2.py:1018]       + DetectionBoxes_Precision/mAP (large): 0.861276
```

3. Recall and Average Recall over small medium and large objects :

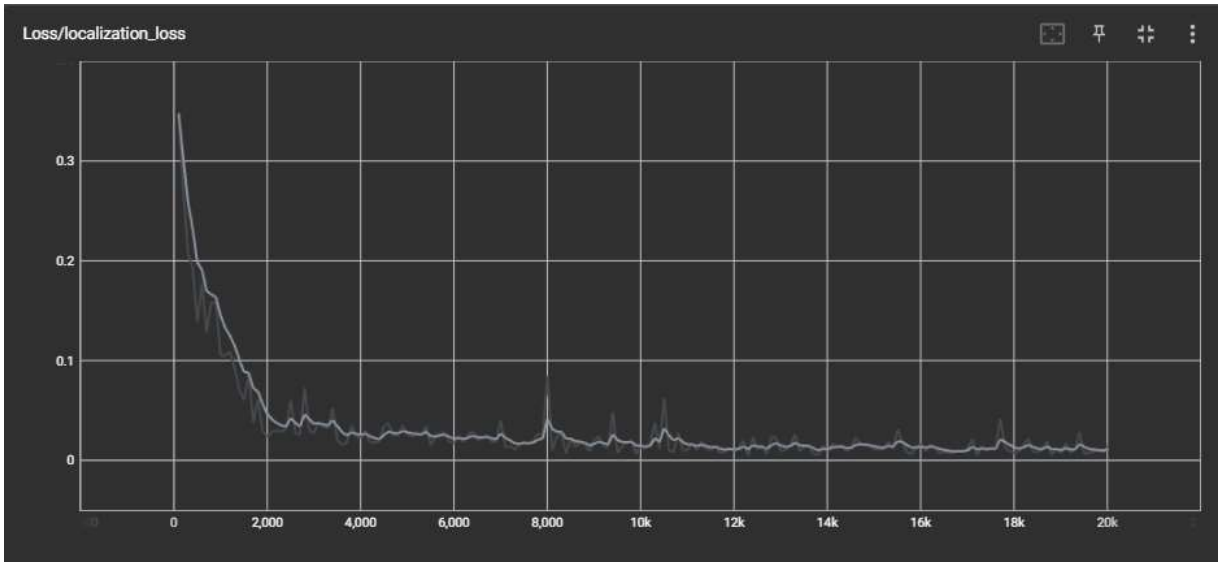
```
INFO:tensorflow:        + DetectionBoxes_Recall/AR@1: 0.875000
I0628 21:17:49.915526 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@1: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@10: 0.875000
I0628 21:17:49.917667 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@10: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100: 0.875000
I0628 21:17:49.918664 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100: 0.875000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0628 21:17:49.919664 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (medium): 0.843750
I0628 21:17:49.921681 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (medium): 0.843750
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (large): 0.866667
I0628 21:17:50.044589 17428 model_lib_v2.py:1018]       + DetectionBoxes_Recall/AR@100 (large): 0.866667
```

4. Different types of Loss functions :

```
INFO:tensorflow:        + Loss/localization_loss: 0.029558
I0628 21:17:50.046946 17428 model_lib_v2.py:1018]       + Loss/localization_loss: 0.029558
INFO:tensorflow:        + Loss/classification_loss: 0.078346
I0628 21:17:50.047945 17428 model_lib_v2.py:1018]       + Loss/classification_loss: 0.078346
INFO:tensorflow:        + Loss/regularization_loss: 0.067200
I0628 21:17:50.049921 17428 model_lib_v2.py:1018]       + Loss/regularization_loss: 0.067200
INFO:tensorflow:        + Loss/total_loss: 0.175104
```

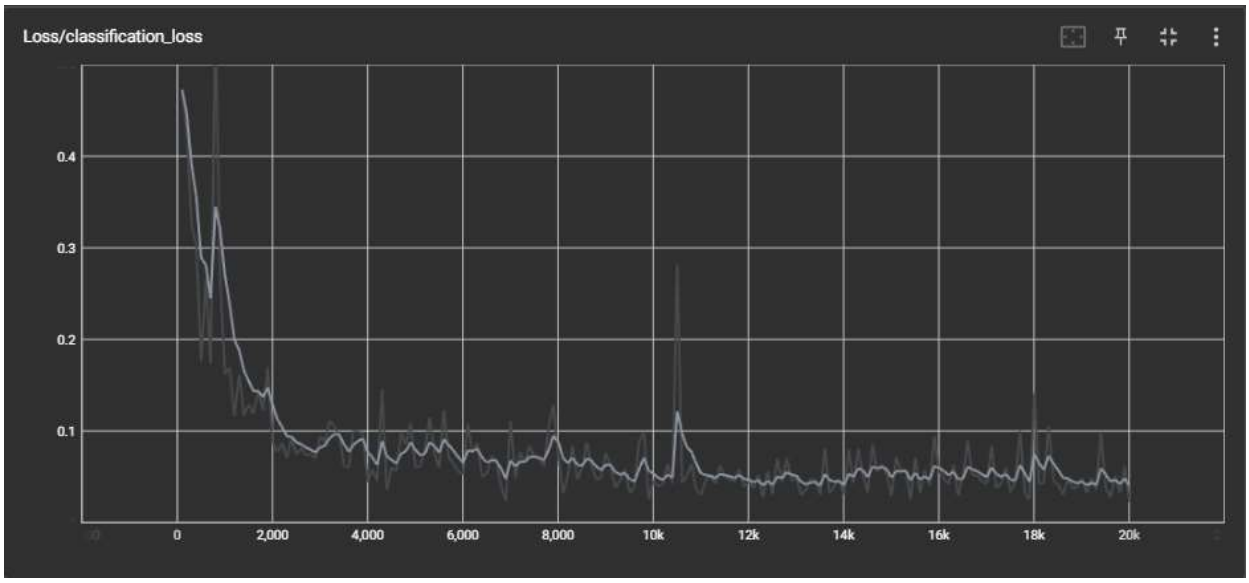
1) Localization Loss (From TensorBoard)

X-Axis = Training Steps
Y-Axis = Loss Value



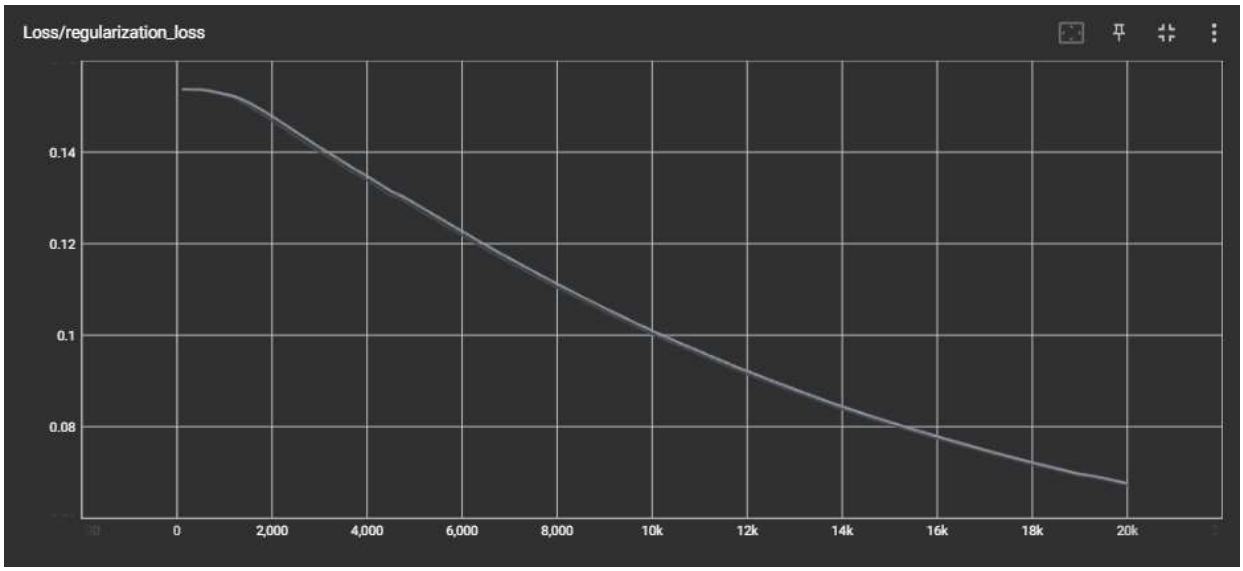
2) Classification Loss (From TensorBoard)

X-Axis = Training Steps
Y-Axis = Loss Value

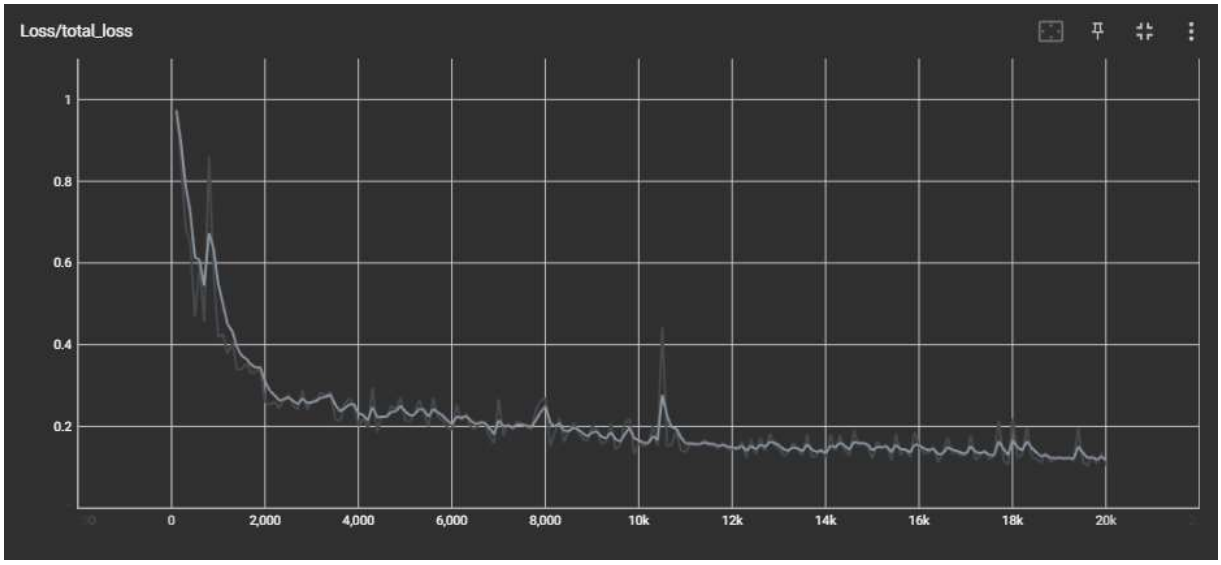


3) Regularization Loss (From TensorBoard)

X-Axis = Training Steps
Y-Axis = Loss Value

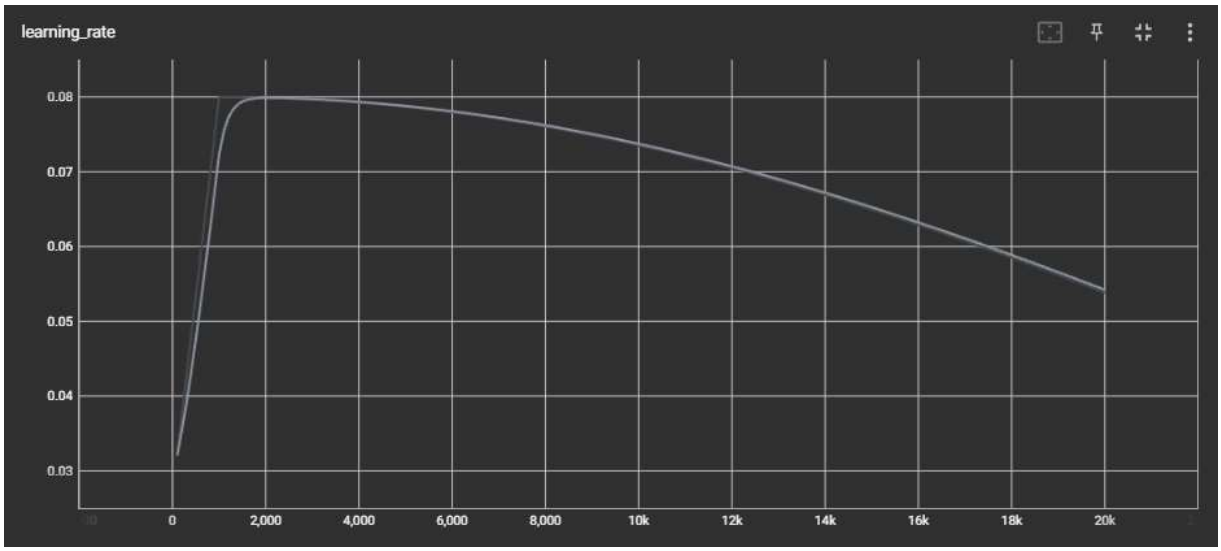


4) Total Loss (From TensorBoard)



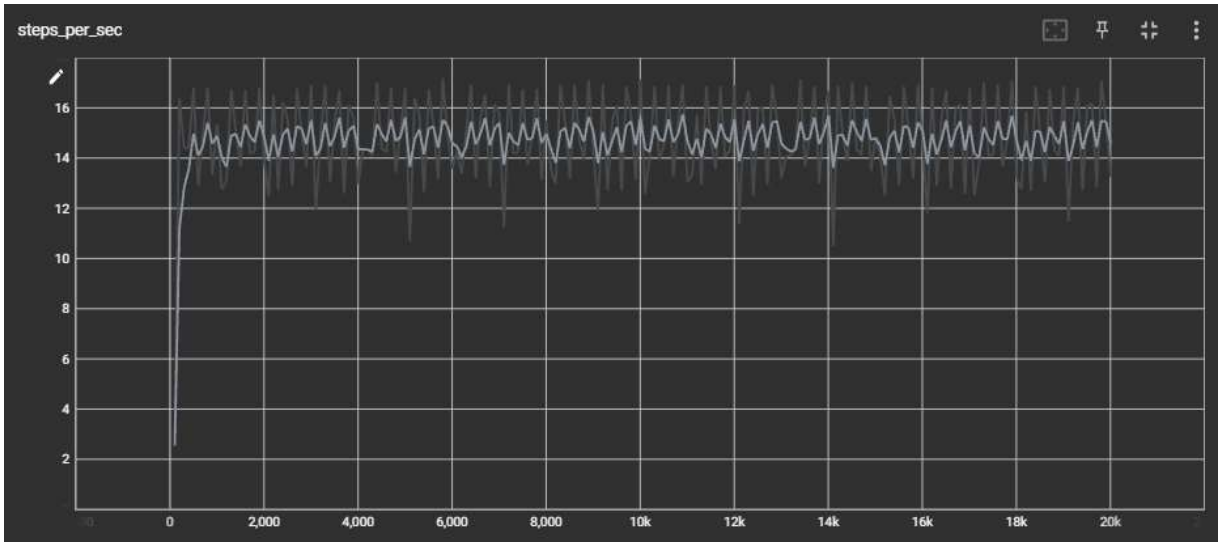
5. Learning rate

Learning rate is a hyper parameter that determines the step size or rate at which a machine learning model adapts or updates its internal parameters during training. It controls the magnitude of adjustments made to the model's weights and biases in response to the calculated gradients of the loss function.

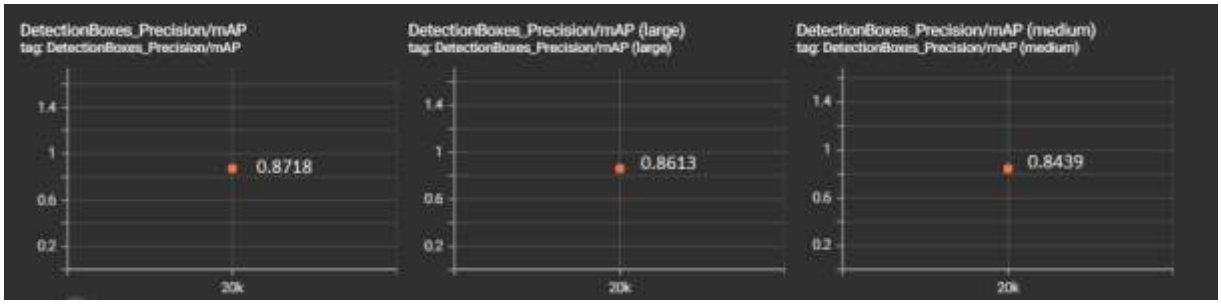


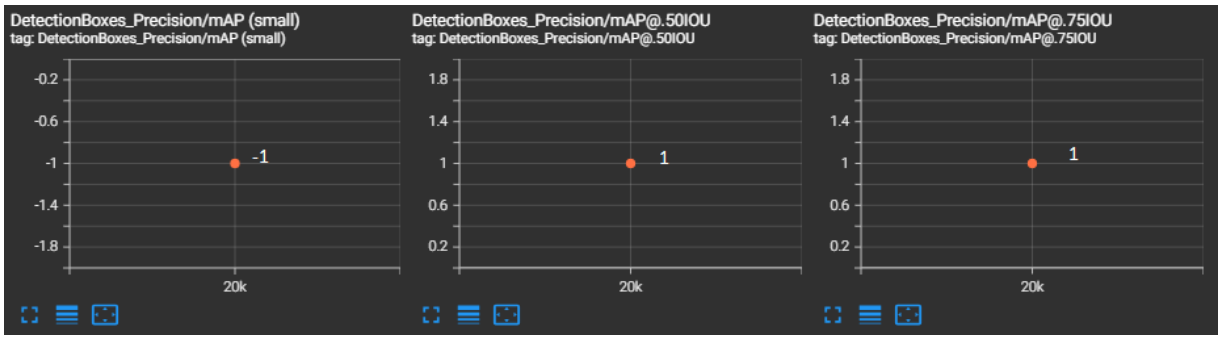
6. Steps per second

Shows how many steps were covered per second while training.

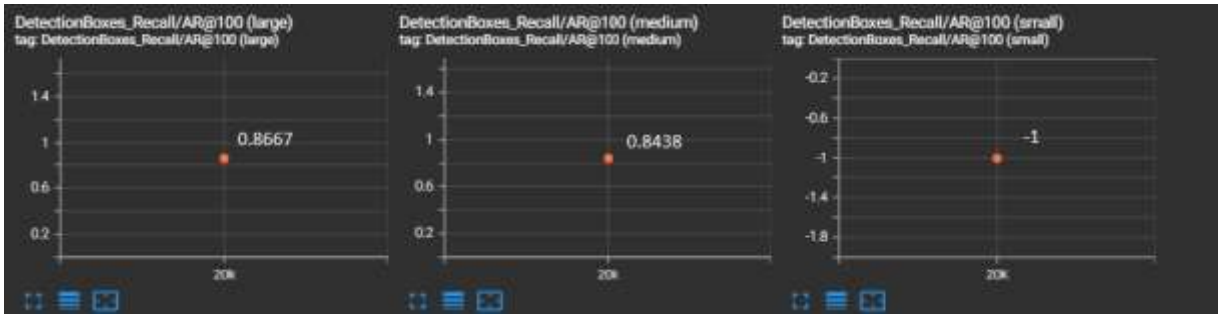


7. Precision of the detection boxes





8. Recall of the Detection Boxes



CHAPTER 7: CONCLUSION

7.1 Conclusion

Sign Language Detection using OpenCV holds immense potential for improving communication and inclusivity for the sign language community. The existing work in this field has demonstrated promising results, but there are still several areas for future exploration and improvement.

In recent years, the field of sign language detection has witnessed significant advancements due to the integration of computer vision techniques, machine learning algorithms, and deep learning models. The combination of Python, OpenCV, and TensorFlow has proven to be a powerful framework for developing robust and efficient sign language detection systems. However, there are still several avenues for further research and development to enhance the capabilities and impact of these systems.

One important area for improvement is the accuracy of sign language detection algorithms. While existing models have demonstrated impressive performance, there is always room for refinement. Fine-tuning the model architectures, exploring advanced deep learning techniques, and leveraging larger and more diverse datasets can contribute to further accuracy gains. Additionally, investigating techniques for handling challenging scenarios such as occlusions, partial visibility of hands, and variations in lighting conditions would make the system more robust and reliable in real-world scenarios.

Expanding the range of recognized gestures is another key objective. Sign languages exhibit a wide range of expressions, including intricate hand movements, facial expressions, and body postures. To achieve comprehensive sign language detection, it is crucial to develop algorithms that can effectively capture and interpret these nuances. This could involve incorporating additional modalities such as facial recognition, pose estimation, and gaze tracking to capture the holistic nature of sign language communication.

Real-time performance on resource-constrained devices is also a significant consideration. While the current systems can achieve real-time performance on high-end machines, adapting them to low-power devices, such as smartphones or embedded systems, is essential for widespread deployment. This requires optimizing the algorithms, reducing computational complexity, and exploring hardware acceleration techniques to ensure efficient execution on these platforms.

In addition to technical improvements, there are broader considerations for enhancing the usability and effectiveness of sign language detection systems. Adapting the systems to different sign languages is crucial, as sign languages exhibit significant variations across different regions and cultures. Customizing the algorithms to recognize specific sign language vocabularies and gestures can greatly improve the usability and accuracy of the systems for users of different sign languages.

Integrating sign language detection systems with natural language processing (NLP) can further enhance their functionality. By combining the recognized sign language gestures with NLP algorithms, the systems can provide real-time translation of sign language into spoken or written language. This integration would enable seamless communication between sign language users and individuals who do not understand sign language, breaking down barriers and promoting inclusivity.

Considering user interaction and feedback is also vital for designing effective sign language detection systems. User interfaces should be intuitive, user-friendly, and accessible, allowing users to easily interact with the system using their preferred input methods, such as a webcam or pre-recorded videos. Additionally, incorporating user feedback mechanisms and iterative design processes can help refine the system based on real-world user experiences and requirements.

The future work in sign language detection using OpenCV has the potential to make a significant impact on the lives of individuals in the sign language community, contributing to a more inclusive and accessible society for all.

In conclusion, the Sign Language Detection project showcases the effectiveness of combining Python, OpenCV, and TensorFlow to create a robust and accurate system for interpreting sign language gestures in real-time. The technology holds significant promise in promoting inclusivity and accessibility for the deaf and hard-of-hearing communities, potentially bridging communication gaps and enriching their lives. As the project continues to evolve and mature, we hope it will become a valuable tool to empower individuals with hearing impairments and contribute to a more inclusive and understanding society.

7.2 Limitations

Despite its potential, sign language detection using OpenCV also faces several limitations. Firstly, the accuracy of the system heavily relies on the quality of the input video feed, making it susceptible to variations in lighting conditions, camera angles, and background clutter. These factors can lead to false detections or missed gestures, affecting the overall reliability of the system.

Variability in Hand Shapes and Positions: Hand sign gestures can vary significantly between individuals due to factors like hand shapes, sizes, and orientations. This variability poses a challenge for the model to accurately detect and classify different hand signs. The model needs to be robust enough to handle variations in hand appearance and adapt to different hand configurations.

Occlusions and Partial Visibility: In real-world scenarios, hand signs can be partially occluded by other objects or body parts, such as clothing, accessories, or the signer's own body. Occlusions can hinder the model's ability to accurately detect and recognize the hand signs. Dealing with partial visibility and occlusions requires advanced techniques such as context-based reasoning or multi-view fusion to infer the sign gesture from the available visual cues.

Real-time Performance: Real-time performance is crucial for practical applications of hand sign recognition. However, achieving real-time inference on resource-constrained devices, such as smartphones or embedded systems, can be challenging due to the computational demands of object detection algorithms. Optimizing the model architecture, leveraging hardware acceleration, or exploring lightweight architectures tailored for hand sign detection can help address this limitation.

Environmental Factors: The performance of the model may be affected by various environmental factors, such as changes in lighting conditions, background clutter, and variations in camera perspectives. Adapting the model to handle such variations and making it robust to different environmental settings is essential to ensure reliable performance across diverse scenarios.

Limited Generalization to Unseen Gestures: While a well-trained model can accurately recognize the gestures present in the training dataset, its ability to generalize to unseen or novel gestures may be limited. The model may struggle to classify new signs or variations that were not present during training. Continual training and adaptation techniques, as well as incorporating transfer learning approaches, can help improve the model's generalization capabilities.

Cultural and Regional Variations: Sign languages can exhibit significant variations across different cultures and regions. Different sign languages may have unique hand configurations, movements, and meanings. Training a model to recognize a specific sign language may not be directly applicable to other sign languages. Adapting the model to handle multiple sign languages or developing language-specific models can help overcome this limitation.

Addressing these limitations requires continued research and development in the field of hand sign recognition. By overcoming these challenges, object detection models can contribute to the development of effective assistive technologies that promote communication, inclusion, and accessibility for deaf and dumb individuals.

7.3 Future Scope of the Project

In future work, there are several aspects that can be further developed and implemented to enhance the proposed hand sign recognition system for the deaf and dumb community. One potential avenue for improvement is to utilize Raspberry Pi as the platform for the system. Raspberry Pi offers a cost-effective and compact solution, making it suitable for portable and embedded applications. By optimizing the system to run on Raspberry Pi, it can be made more accessible and convenient for everyday use.

Improving the image processing capabilities is another crucial aspect. Enhancing the image processing algorithms can enable the system to facilitate bi-directional communication. In addition to recognizing sign language gestures and converting them into text or speech, the system can also interpret spoken language or text and translate it into sign language gestures. This bidirectional functionality would allow seamless communication between individuals using sign language and those who are not familiar with it.

Expanding the system's ability to recognize signs that involve motion is an important area of research. While the current system focuses on static hand signs, incorporating dynamic gestures would further enrich its capabilities. This could involve capturing and analyzing the temporal aspects of hand movements to recognize more complex sign language expressions. The integration of motion detection and tracking algorithms could enable the system to interpret dynamic gestures accurately.

Converting the sequence of gestures into meaningful text, words, and sentences is another direction for improvement. Currently, the system recognizes individual hand signs, but extending its functionality to comprehend the sequential nature of sign language would enhance its usability. By considering the context and order of gestures, the system can generate accurate textual representations of complete phrases or sentences. This would greatly benefit users who rely on sign language for their communication needs.

Transforming the recognized text into audible speech would be a valuable addition to the system. By integrating text-to-speech synthesis techniques, the system can provide an audio output corresponding to the detected sign language. This would allow individuals who are not familiar with sign language to understand and respond to the communication effectively. The combination of visual recognition and audio synthesis would create a more inclusive and versatile communication experience.

In summary, future work on the proposed hand sign recognition system can focus on utilizing Raspberry Pi as the platform, enhancing image processing capabilities, enabling bi-directional communication, recognizing motion-based signs, converting sequential gestures into text and speech, and ultimately providing a user-friendly and accessible solution for the deaf and dumb community. The integration of deep learning techniques and the system's ability to operate with standard cameras and minimal equipment make it a promising technology for facilitating effective communication and fostering inclusivity.

I. References

- Cruz-Albarran, I., Nakano, Y. I., & Kakusho, K. (2021). Real-time sign language recognition with Convolutional Neural Networks and OpenCV. In 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 4284-4289). IEEE.

- Dey, S., Chaudhury, S., & Singh, K. (2020). Real-time hand gesture recognition for sign language interpretation using OpenCV. In 2020 5th International Conference on Computing, Communication and Security (ICCCS) (pp. 1-6). IEEE.

- Priyanka, V., Muthusamy, N., & Anbuselvi, M. (2021). Real-time sign language recognition using TensorFlow and OpenCV. In 2021 International Conference on Electronics and Sustainable Communication Systems (ICESC) (pp. 138-141). IEEE.

- Yang, C., Bai, B., Wang, L., Li, L., Zhang, J., & Ji, Z. (2020). Real-time American Sign Language Recognition Using Deep Learning and Hand Segmentation. IEEE Access, 8, 84395-84404.

- Harpreet, K., & Agrawal, A. (2020). Real-time American Sign Language Recognition using deep learning approach. In 2020 2nd International Conference on Advances in Science & Technology (ICAST) (pp. 1-5). IEEE.

- W. Li, H. Pu and R. Wang, "Sign Language Recognition Based on Computer Vision," 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 2021, pp. 919-922, doi: 10.1109/ICAICA52286.2021.9498024.

- S. Shrenika and M. Madhu Bala, "Sign Language Recognition Using Template Matching Technique," 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 2020, pp. 1-5, doi: 10.1109/ICCSEA49143.2020.9132899.

- S. Ikram and N. Dhanda, "American Sign Language Recognition using Convolutional Neural Network," 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), Kuala Lumpur, Malaysia, 2021, pp. 1-12, doi: 10.1109/GUCON50781.2021.9573782.

- N. Kumar and A. K. Singh Bisht, "Hand sign detection using deep learning single shot detection technique," 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-7, doi: 10.1109/ViTECoN58111.2023.10157550.

- A. Pardasani, A. K. Sharma, S. Banerjee, V. Garg and D. S. Roy, "Enhancing the Ability to Communicate by Synthesizing American Sign Language using Image Recognition in A Chatbot for Differently Abled," 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2018, pp. 529-532, doi: 10.1109/ICRITO.2018.8748590. Ruchi Manish Gurav and Premanand K Kadbe. Real time finger tracking and contour detection for gesture recognition using opencv. In 2015 International Conference on Industrial Instrumentation and Control (ICIC), pages 974–977. IEEE, 2015.

- Malavika Suresh, Avigyan Sinha, and RP Aneesh. Real-time hand gesture recognition using deep learning. International Journal of Innovations and Implementations in Engineering, 1, 2019.

- Parshwa P Patil, Maithili J Phatak, Suharsh S Kale, Premjeet N Patil, Pranav S Harole, and CS Shinde. Hand gesture recognition for deaf and dumb. Hand, 6(11), 2019

- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9). (pp. 1-9). IEEE.

- Omkar Vedak, Prasad Zavre, Abhijeet Todkar, and Manoj Patil. Sign language interpreter using image processing and machine learning. International Research Journal of Engineering and Technology (IRJET), 2019.

II. Websites

- https://www.youtube.com/
- https://machinelearningmastery.com/what-is-deep-learning/
- https://www.tensorflow.org/
- https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md (TensorFlow Model Zoo)
- https://github.com/tzutalin/labelImg (Labelling the images)
- https://www.researchgate.net/publication/331421347
- https://www.mygreatlearning.com/blog/object-detection-usingtensorflow/
- https://gilberttanner.com/blog/creating-your-own-objectdetector/
- https://medium.com/@nnamaka7/multi-object-detection-with-thetensorflow-object-detection-api-afb3d14e036c
- https://www.w3schools.blog/python-opencv
- https://www.geeksforgeeks.org/python-opencv-cv2-imshow-method/
- Chat GPT - https://chat.openai.com/
- https://pyimagesearch.com/2020/06/22/turning-any-cnn-imageclassifier-into-an-object-detector-with-keras-tensorflow-and-opencv/
- https://neptune.ai/blog/how-to-train-your-own-object-detector-usingtensorflow-object-detection-api

III. Summary

We have developed a sign language detection system using TensorFlow and OpenCV. We begin by installing and setting up the necessary components, including the TensorFlow Object Detection API, which simplifies the process of building an object detection model. We collect images using a webcam and label them by marking the hand signs of interest with bounding boxes using the labelIMG repository.

The labeled images serve as training data. Annotations are saved as XML files in the PASCAL VOC format. We select the SSD MobileNet V2 FPNLite 320x320 model from the TensorFlow Model Zoo and train it using the labelled images. We create a label map and generate a TFRecords file for efficient training. The pipeline.config file is customized to meet our requirements. The training process involves feeding the labeled images and labels to the model to improve its ability to detect hand signs accurately. Once training is complete, the model can be used for real-time detection of hand signs in input images or video frames.

Our systematic approach establishes a robust object detection pipeline for accurate identification of hand signs using our custom-trained model.

IV. Further Readings

- N. Kumar and A. K. Singh Bisht, "Hand sign detection using deep learning single shot detection technique," 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-7, doi: 10.1109/ViTECoN58111.2023.10157550.

- Yang, C., Bai, B., Wang, L., Li, L., Zhang, J., & Ji, Z. (2020). Real-time American Sign Language Recognition Using Deep Learning and Hand Segmentation. IEEE Access, 8, 84395-84404.

- Dey, S., Chaudhury, S., & Singh, K. (2020). Real-time hand gesture recognition for sign language interpretation using OpenCV. In 2020 5th International Conference on Computing, Communication and Security (ICCCS) (pp. 1-6). IEEE.