Project Report

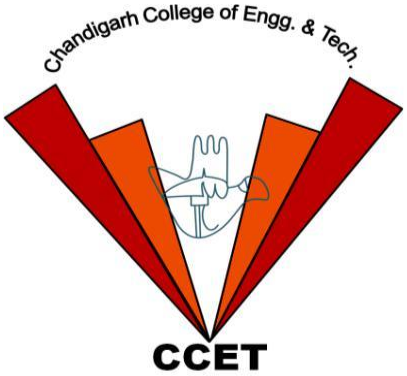
On

REAL-TIME SIGN LANGUAGE DETECTION APPLICATION

A Project Report submitted in partial fulfilment of the requirements for the award of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Under the supervision of
Dr. Sarabjeet Singh



CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY
(DEGREE WING)
Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University, Chandigarh
Sector-26, Chandigarh. PIN-160019
JAN-JUNE, 2022

Submitted by

ABHISHEK PANT(CO18305)
KANISH CHAUHAN(CO18330)
SHUVAM ROY (CO18349)

CSE- 8th SEM

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University, Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No**. :** 0172-2750872

Department of Computer Sc. & Engineering

STUDENTS DECLARATION

I hereby declare that the work presented in this report entitled "**Real-Time Sign Language Detection Application**", in fulfilment of the requirement for the award of the degree Bachelor of Engineering in Computer Science & Engineering, submitted in CSE Department, Chandigarh College of Engineering & Technology (Degree wing) affiliated to Punjab University, Chandigarh, is an authentic record of my own work carried out during my degree under the guidance of **Dr. Sarabjeet Singh**. The work reported in this has not been submitted by me for award of any other degree .

SUBMITTED BY
ABHISHEK PANT (CO18305)
KANISH CHAUHAN (CO18330)
SHUVAM ROY (CO18349)
CSE 8th Semester

ACKNOWLEDGEMENT

Perseverance, inspiration and motivation have played a great role in the success of any venture. It would be incomplete to submit this report without acknowledging the people behind this endeavour and without whose support we wouldn't have able to achieve this. We would like to thank **Dr. Sarabjeet Singh** for giving me an opportunity to work on such a fabulous project and for their able guidance and support in completing my project. Secondly we would like to thank my parents for their constant support which helped me finish our report in limited time. It gives us immense pleasure to express our gratitude to everyone who shared with me their precious time and effort during the project.

ABSTRACT

Hand gesture recognition is a very useful technological advancement as an alternative user interface for providing real-time data to a computer. Classical interaction tools like mouse, keyboard limit the way we interact with our system. Also, this can be very helpful for the deaf and dumb people in interacting and communicating with other people and also computer systems. It can also be used as a sign language translator for the deaf and dumb people. The aim of this project is thus to create a sign recognition detector using Deep Learning and CNN that can detect basic patterns like numbers or signs based on the trained data set being used. Existing gesture recognition software make use of a hardware system design with motion sensors that are not efficient. By implementing a CNN model using Tensor flow, this system aims to make use of just the device camera to live capture and detect the sign automatically. The data set can be freshly trained based on the user requirement. The future scope of the project will be to train the model to store and convert the text into program input or voice which may require more complex algorithms and understanding of Neural Networks and Deep Learning Technologies. Throughout the long term, correspondence has played an indispensable job in return of data and sentiments in one's day to day existence. Sign language is the main medium through which deaf and mute individuals can interact with rest of the world through various hand motions. With the advances in machine learning, it is possible to detect sign language in real time. We have utilized the OpenCV python library, Tensorflow Object Detection pipeline and transfer learning to train a deep learning model that detects sign languages in Real time

Keywords—Hand gesture recognition, Deep Learning, Tensorflow, sign language, CNN model American Sign Language (ASL), Sign Language Detection, Transfer learning, OpenCV

INDEX

1. INTRODUCTION

The sign language is a very important way of communication for deaf-dumb people. In sign language each gesture has a specific meaning. Sign language is a gesture-based language for communication of deaf and dumb people. It is a non-verbal language used by them to communicate more effectively with other people. Unfortunately, there are not many technologies which help in connecting this social group to the rest of the world. Understanding sign language is one of the primary enablers in helping users of sign language communicate with the rest of the society. The disabled are the main users of sign language, and just a few others, such as families, campaigners, and teachers, comprehend it. The natural cue is a manual (hand-handed) expression agreed upon by the user (conventionally), recognised as limited in a certain group (esoteric), and utilised by a deaf person as a substitute for words (as opposed to body language). A formal gesture is a cue that is established consciously and has thesame language structure as the spoken language of the society. More than 360 million of world population suffers from hearing and speech impairments. Sign language detection is a project implementation for designing a model in which web camera is used for capturing images of hand gestures which is done by open cv. After capturing images, labelling of images are required and then pre trained model SSD Mobile net v2 is used for sign recognition. Thus, an effective path of communication is developed between deaf and normal audience.

Three steps must be completed in real time to solve our problem:

1. Obtaining footage of the user signing is step one (input).
2. Classifying each frame in the video to a sign.
3. Reconstructing and displaying the most likely Sign from classification scores (output).

People with hearing impairments are left behind in online conferences, office sessions, schools. They usually use basic text chat to converse — a method less than optimal. With the growing adoption of telehealth, deaf people need to be able to communicate naturally with their healthcare network, colleagues and peers regardless of whether the second person knows sign language. Being able to achieve a uniform sign language translation machine is not a simple task, however, there are two common methods used to address this problem namely sensor based sign language recognition and Vision-based sign language recognition. Sensor based sign language recognition uses designs such as the robotic arm with a sensor, smart glove, golden glove for the conversion of ASL Sign language to speech. But the issue

is that many people do not use it. Also, one must spend money to purchase such a glove, which is not easily available.

1.1. BACKGROUND

Some of the major problems faced by a person who are unable to speak is they cannot express their emotion as freely in this world. Utilize that voice recognition and voice search systems in smartphone(s). Audio results cannot be retrieved. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc. because all those apps are based on voice controlling.

There is a need for such platforms for such kind of people. American Sign Language (ASL) is a complete, complex language that employs signs made by moving the hands combined with facial expressions and postures of the body. It is the go-to language of many North Americans who are not able to talk and is one of various communication alternatives used by people who are deaf or hard-of-hearing.

While sign language is very essential for deaf-mute people, to communicate both with normal people and with themselves, is still getting less attention from the normal people. The importance of sign language has been tending to ignored, unless there are areas of concern with individuals who are deaf-mute. One of the solutions to talk with the deaf-mute people is by using the mechanisms of sign language.

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people. Various sign language systems have been developed by many manufacturers around the world but they are neither flexible nor cost-effective for the end users.

1.2. PROBLEM STATEMENT

Given a hand gesture, implementing such an application which detects pre-defined American sign language (ASL) in a real time through hand gestures and providing facility for the user to be able to store the result of the character detected in a txt file, also allowing such users to build their customized gesture so that the problems faced by persons who aren’t able to talk vocally can be accommodated with technological assistance and the barrier of expressing can be overshadowed.

2. MOTIVATION OF WORK

Communication is one of the basic requirements for survival in society. Deaf and dumb people communicate among themselves using sign language but normal people find it difficult to understand their language. Sign language recognition is a problem that has been addressed in research for years. However, we are still far from finding a complete solution available in our society. Also, lack of datasets along with variance in sign language with locality has resulted in restrained efforts in such hand sign gesture detection. Our project aims at taking the basic step in bridging the communication gap between normal people and deaf and dumb people using a simple and effective deep learning vision based model to recognize sign languages in real time. Effective extension of this project to words and common expressions may not only make the deaf and dumb people communicate faster and easier with the outer world, but also provide a boost in developing autonomous systems for understanding and aiding them.

2.1. PROPOSED SYSTEM

1) Collect images for deep learning using webcam and OpenCV
2) Label images for sign language detection using Labelmg.
3) Setup Tensor flow Object Detection pipeline configuration
4) Use transfer learning to train a deep learning model
5) Detect sign language in real time using OpenCV

In this project, we have first collected images using webcam and OpenCV. After collecting the images, the second task is labelling those images for Sign Language Detection. We labeled the collected images using LabelImg. After labelling the images, it is mandatory to set up Tensorflow Object Detection pipeline configuration. Now, after setting up the Tensorflow Object Detection configuration, we will use Transfer Learning to train a deep learning model. Now, we are able to detect sign language in real time using OpenCV

2.2. OBJECTIVES

One of the solutions to communicate with the deaf-mute people is by using the services of sign language interpreter. But the usage of sign language interpreters could be expensive. Cost-effective solution is required so that the deaf-mute and normal people can communicate normally and easily.

Our strategy involves implementing such an application which detects pre-defined American sign language (ASL) through hand gestures. For the detection of movement of gesture, we would use basic level of hardware component like camera and interfacing is required. Our application would be a comprehensive User-friendly Based system built on PyQt5 module. Instead of using technology like gloves or kinect, we are trying to solve this problem using state of the art computer vision and machine learning algorithms.

This application will comprise of two core module one is that simply detects the gesture and displays appropriate alphabet. The second is after a certain amount of interval period the scanned frame would be stored into buffer so that a string of character could be generated forming a meaningful word.

Additionally, an-add-on facility for the user would be available where a user can build their own custom-based gesture for a special character like period (.) or any delimiter so that a user could form a whole bunch of sentences enhancing this into paragraph and likewise. Whatever the predicted outcome was, it would be stored into a .txt file.

2.3. LIMITATIONS

Before we start on with the limitation part, we will first get familiar with the Current System on which we will be working on. Currently the main sign language recognition approach used is sensor-based sign detection. But sensor-based devices are not convenient as hand gloves, helmet etc. are required by the user. Hardware devices like kinetic sensors (by Microsoft) develops a 3D model of the hand and observes the hand movements and their orientations. A glove-based approach was another technique wherein the user was required to wear aspecial glove that recognized the position and orientation of the hand.

Limitations In Existing System:

- High initial setup cost and less practical feasibility
- User requires adequate knowledge of technology to use the hardware system
- Considerable E-waste due to large number of sensors used

Our statement of contribution in this project report, is to develop a system that can effectively recognize sign language using deep learning techniques. The proposed system intends to help computers recognize sign language,which could then be interpreted by other people. Convolutional neural networks have been employed to recognize sign language gestures. The image dataset used consists of dynamic sign language gestures captured on a system attached camera. Preprocessing was performed on the images, which then served as the cleaned input. The results are obtained by retraining and testing this sign language gestures dataset on a convolutional neural network model.

3. SYSTEM REQUIREMENTS

3.1.1. HARDWARE REQUIREMENTS

- Intel Core i3 3rd gen processor or later.
- 512 MB disk space.
- 512 MB RAM.
- Any external or inbuild camera with minimum pixel resolution 200 x 200 (300ppi or 150lpi) 4-megapixel cameras and up.

3.1.2. SOFTWARE REQUIREMENTS

- Microsoft Windows XP or later / Ubuntu 12.0 LTS or later /MAC OS 10.1 or later.
- Python Interpreter (3.6).
- TensorFlow framework, Keras API.
- PyQT5, Tkinter module.
- Python OpenCV2, scipy, qimage2ndarray, winGuiAuto, pypiwin32, sys, keyboard, pyttsx3, pillow libraries.

4. METHODOLOGY

Sign language is a major form of communication used by almost 70 million people throughout the world.The problem arises as there is no standard mode of signlanguage practised. Different people use different signsin different places. Hence, a a methodical solution that can train itself based on one-time user input is thefoundation of the research. The experimentive datasetis a primary dataset collected by recording the images of certain basic signs used worldwide. On the collecteddataset, we divided our approach to tackle the classification problem into three stages.

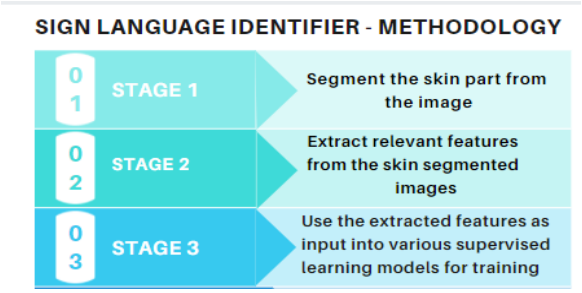


Fig. 1 : Stages of Implemented Methodology

The first stage is to segment the skin part from the image, as the remaining part can be regarded as noise. The second stage is to extract relevant features from the skin segmented images which can prove significant forthe next stage i.e. learning and classification. The third stage as mentioned above is to use the extracted features as input into various supervised learning models for training and then finally use the trained models for classification.

The use of simple machine learning techniques in this research is done to eliminate the ue of sensors or motion capture signal processors by training a model todetect live images based on a set of trained parameters.The research method extends to work on creating a system that allows user to train certain signs at any timeas per user requirement.

ASL recognition is not a new computer vision problem. Over the past two decades, researchers have used classifiers from a variety of categories that we grouped roughly into linear classifiers, neural networks and Bayesian networks. The first approach in relation to sign language recognition was by Bergh in 2011. Haar wavelets and database searching were employed to build a hand gesture recognition system. Although this system gives good results, it only considers six classes of gestures. Many types of research have been carried out on different sign languages from different countries. For example, a BSL recognition model, which understands finger-spelled signs from a video, was built . As Initial, a

histogram of gradients (HOG) was used to recognize letters, and then, the system used hidden Markov models (HMM) to recognize words. In another paper, a system was built to recognize sentences made of 3-5 words. Each word ought to be one of 19 signs in their thesaurus. Hidden Markov models have also been used on extracted features . In 2011, a real time American Sign Language recognition model was proposed utilizing Gabor filter and random forest. A dataset of color and depth images for 24 different alphabets was created. An accuracy of 75% was achieved utilizing both color and complexity images, and 69% using depth images only. Depth images were only used due to changes in the illumination and differences in the skin pigment. In 2013, a multilayered random forest was also used to build a real time ASL model. The system recognizes signs through applying random forest classifiers to the combined angle vector. An accuracy of 90% was achieved by testing one of the training images, and an accuracy of 70% was achieved for a new image. An American Sign Language alphabet recognition system was first built by localizing hand joint gestures using a hierarchical mode seeking and random forest method. An accuracy of 87% was achieved for the training, and accuracy of 57% when testing new images. In 2013, the Karhunen-Loeve Transform was used to classify gesture images of one hand into 10 classes. These were translated and the axes were rotated to distinguish a modern coordinate model by applying edge detection, hand cropping, and skin filter techniques. Another approach is to use deep learning techniques. This approach was used to build a model that recognizes hand gestures in a continual video stream utilizing DBN models. An accuracy of over 99% was achieved. Another research used a deep learning technique, whereby a feed forward neural network was used to classify a sign. Many image pre-processing methods have been used, such as background subtraction, image normalization, image segmentation and contrast adjustment. All the works discussed above depended on the extraction of the hand before it is fed to a network. Kang, Tripathi & Nguyen built a real time sign finger spelling recognition system using CNNs from the depth map. The authors collected 31,000 depth maps with 1,000 images for each class by using the Creative Senz3D camera. They had 31 various hand signs from 5 various persons. They utilized hand segmentation and assumed that the hand had to be near to the camera, which helped to make the bounding boxes in the same input size of 256, and 256. This work utilized only the depth map method by using the (Creative- Senz3D) camera, which is expensive and not available to everyone. Therefore, the proposed system cannot be implemented in a normal PC camera. This project differs from previous works in several ways. First, we have created our own dataset of 557 images. The dataset includes 5 gestures that are “Hello”, “Yes”, “No”, “Thank you” and “I Love

You" and all the alphabets excluding J and Z because they are dynamic in nature. Second, we have created several user-friendly ways like website, QR code for users to access the system in order to communicate effectively.

4.1. DESIGN

4.1.1. SYSTEM ARCHITECTURE

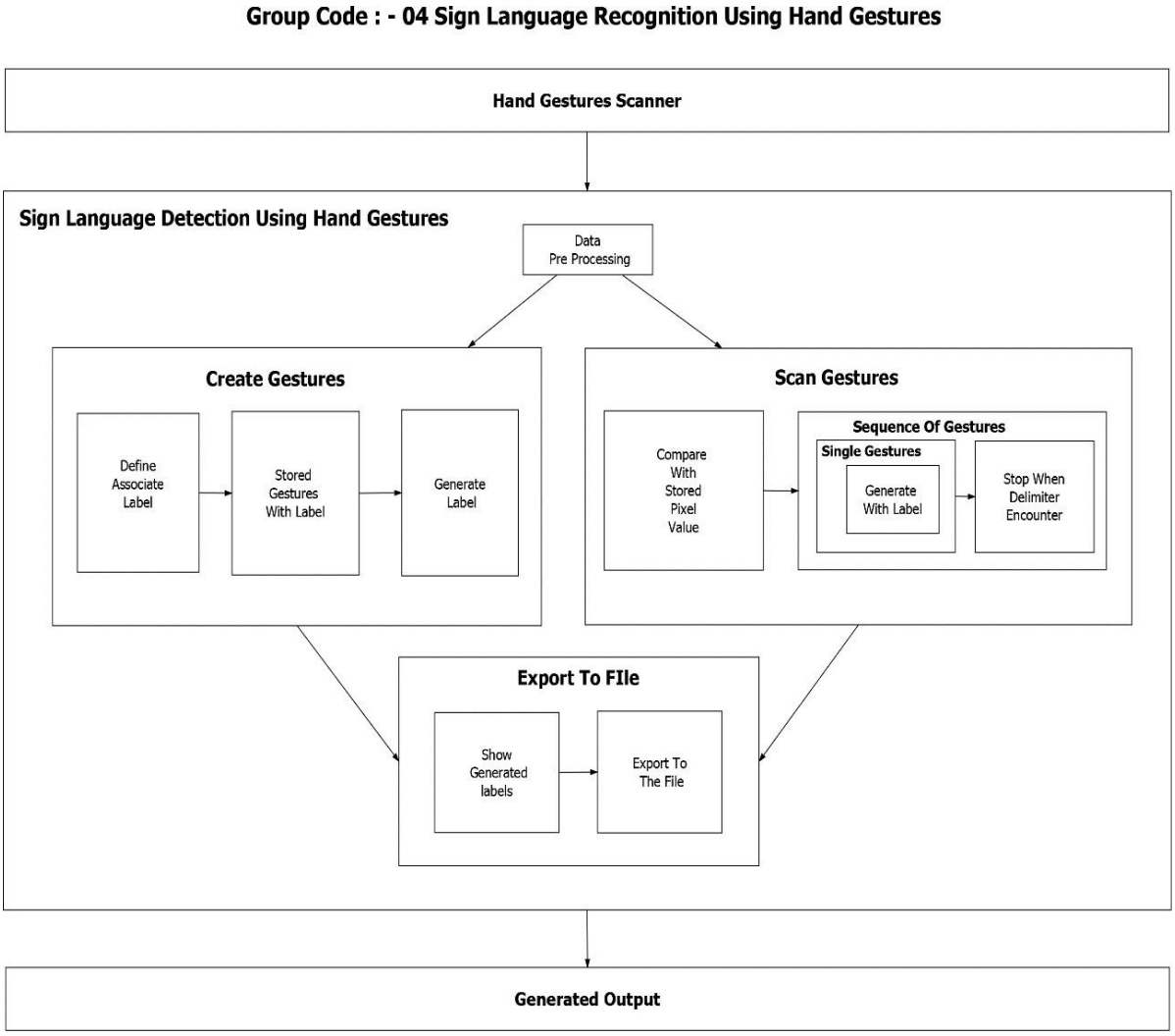


Fig 2: System Architecture for Sign Language Recognition Using Hand Gestures.

4.1.2. MODULES IN THE SYSTEM

- **Data Pre-Processing** – In this module, based on the object detected in front of the camera its binary images is being populated. Meaning the object will be filled with solid white and background will be filled with solid black. Based on the pixel's regions, their numerical value in range of either 0 or 1 is being given to next process for modules.
- **Scan Single Gesture** – A gesture scanner will be available in front of the end user where the user will have to do a hand gesture. Based on Pre-Processed module output, a user

shall be able to see associated label assigned for each hand gestures, based on the predefined American Sign Language (ASL) standard inside the output window screen.

- **Create gesture** –A user will give a desired hand gesture as an input to the system with the text box available at the bottom of the screen where the user needs to type whatever he/she desires to associate that gesture with. This customize gesture will then be stored for future purposes and will be detected in the upcoming time.
- **Formation of a sentence** – A user will be able to select a delimiter and until that delimiter is encountered every scanned gesture character will be appended with the previous results forming a stream of meaning-full words and sentences.
- **Exporting** – A user would be able to export the results of the scanned character into an ASCII standard textual file format.

4.1.3. USE CASE DIAGRAM

Group Code: -4 Sign Language Recognition Using Hand Gestures

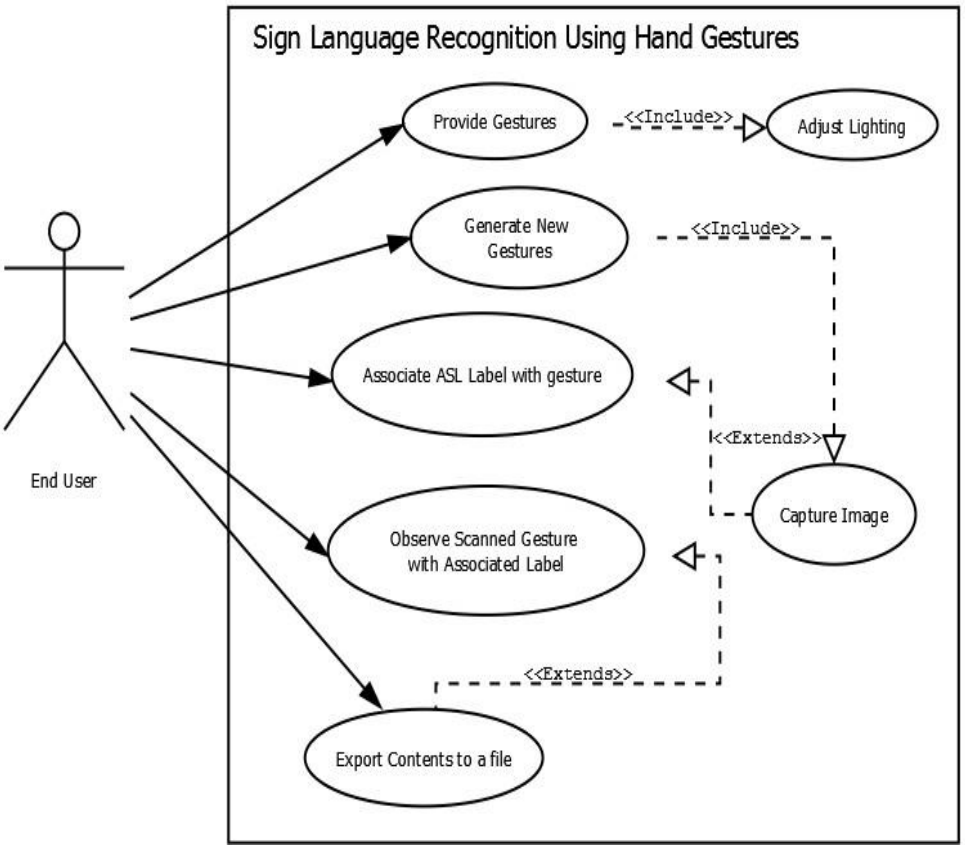


Fig 3: Use Case Diagram for Sign Language Recognition Using Hand Gestures.

4.1.4. ACTIVITY DIAGRAM

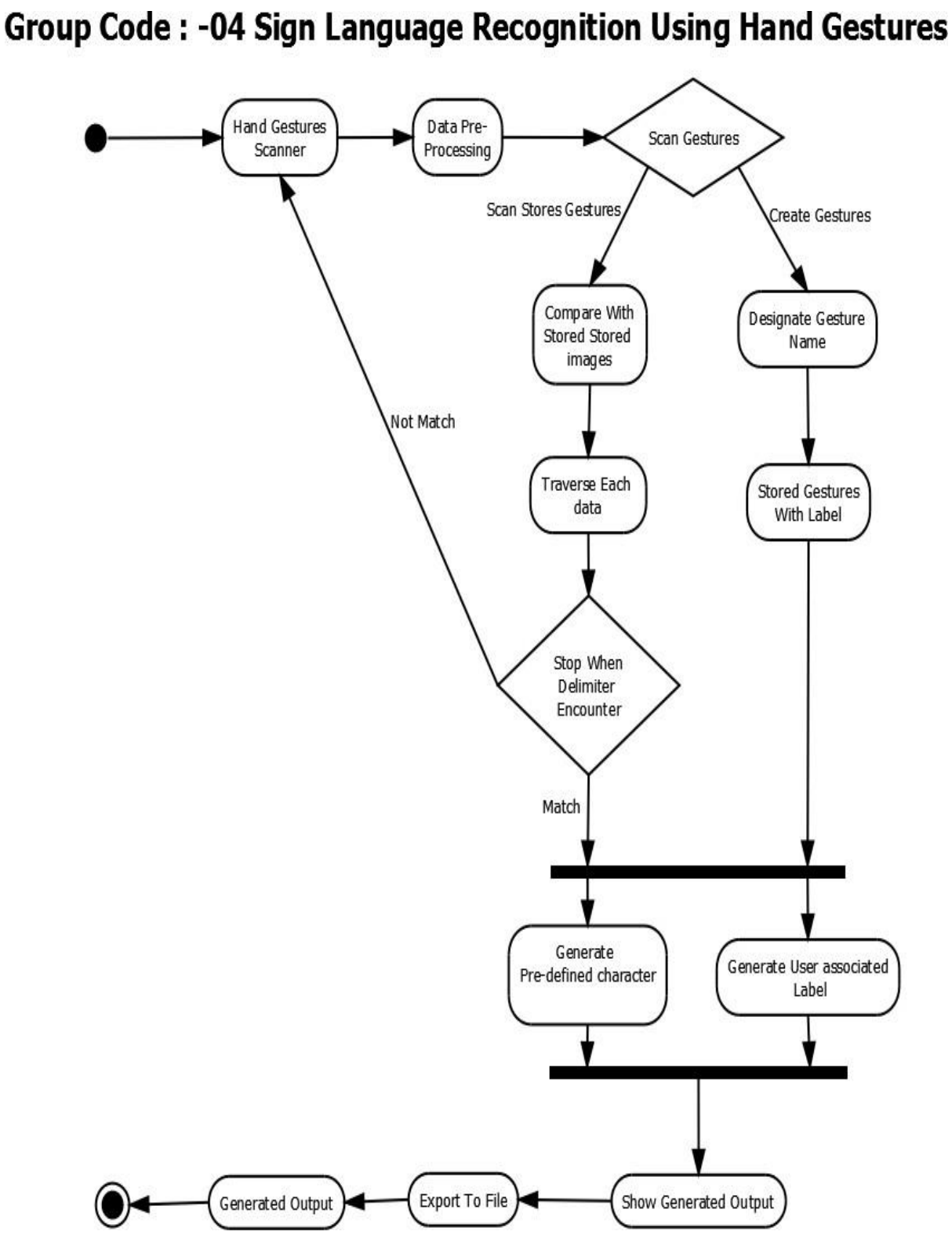


Fig 4: Activity Diagram for Sign Language Recognition Using Hand Gestures.

4.2. IMPLEMENTATION

Techniques like recognising hand motion trajectories for individual signals and segmenting hands from the backdrop to predict and thread them into semantically acceptable and intelligible phrases are employed in sign language recognition. Furthermore, there are challenges in gesture recognition, motion modelling, motion analysis, pattern identification, and machine learning. Handcrafted parameters or parameters that are not manually specified are used in SLR models. The model's ability to categorise is impacted by the model's backdrop and environment, such as the room's illumination and the pace at which the motions are made. Because of the variations in views, the gesture seems unique in 2D space.Systems that recognise gestures are divided into two categories: sensor-based and vision-based systems. Sensor-equipped devices acquire data in the sensor-based method.The trajectory, position, and velocity of the hand are all parameters to consider. Vision-based approaches, on the other hand, leverage graphics from hand gesture video recordings. The phases involved in achieving sign language recognition are as follows: The camera for the sign language recognition system is as follows: The suggested sign language recognition system is based on a web camera collected frame on a laptop or PC. Image processing is carried out with the help of the OpenCV Python computer library. Taking Photographs: Under order to gain improved accuracy through a huge dataset, several photographs of distinct sign language signals were collected from various angles and in variable light conditions.

Segmentation:

After the capturing portion is completed, a specific section from the full image is picked that contains the sign language symbol to be predicted. For the sign to be detected, bounding boxes are used. These boxes should be tightly packed around the picture region to be detected. The hand movements that were labelled were given specific names. The labelling was done with the LabelImg tool. Image selection for training and testing purposes

TF Record Creation:

Multiple training and testing photos were used to produce record files.

There are two types of machine learning methods: supervised and unsupervised. Supervised machine learning is a method of teaching a system to recognise patterns in incoming data so that it predicts future data. Supervised machine learning applies a collection of known training data to labelled training data to infer a function.

Dataset:

For this project, a user defined dataset is used. It is a collection of over 557 images.This dataset contains a total of 5 symbols i.e.,Hello, Yes, No, I Love You and Thank You, which is quite useful while dealing with the real time application.

Convolutional Neural Network:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning system that takes an input picture and give priority (learnable weights and biases) to various aspects/objects while also identifying them. Other classification algorithms need significantly more pre- processing than a ConvNet. ConvNets learns these filters/characteristics with adequate training, whereas simple techniques require filter hand-engineering.

ConvNets are multilayer artificial neural networks that handles input in two dimensions or three dimensions. Every layer in the network is made up of several planes, which can be 2D or 3D, and each plane is made up of a huge number of independent neurons, with layer neurons from neighbouring layers linked but not from the same layer neurons.

A ConvNet captures the Spatial and Temporal aspects of an image by applying appropriate filters. Furthermore, reducing the number of parameters involved and reusing weights resulted in the architecture performing better fitting to the picture collection. ConvNet's major goal is to make image processing easier by extracting relevant characteristics from images while preserving crucial information that is must for making accurate predictions. This is highly

useful for developing an architecture that is not just capable of collecting and learning characteristics but also capable of handling massive volumes of data.

The implementation of the proposed system is done in 5 major steps:

- **STEP1:** A separate sub-directory is created for the IMAGES_PATH where the collected images and their labels will be stored. The labels are initialised and live images are captured and collected using opencv and stored in the respective label folder.
- **STEP2:** Each of these images are labelled using Labellmg tool to graphically label the images and also remove the skin part and unwanted background from the image for training. The attributes and coordinates of the labelled image is stored in an XML file.
- **STEP 3:** Tensorflow Object Detection pipeline configuration is setup. The labels are converted and saved in a pbtxt file that contains text graph definition in protobuf format.
- **STEP 4:** The configurations are pipelined from the config file and trained using the SD Mobilenet model in Object Detection in python. 5000 train steps are used to train the model. Transfer Learning is used to train a deep learning model and load the trained model from checkpoint.
- **STEP 5:** The category boxes are used to visualize and label the boxes. Normalised coordinates are used for the visualised box images.OpenCV is used to detect the sign language in real time.

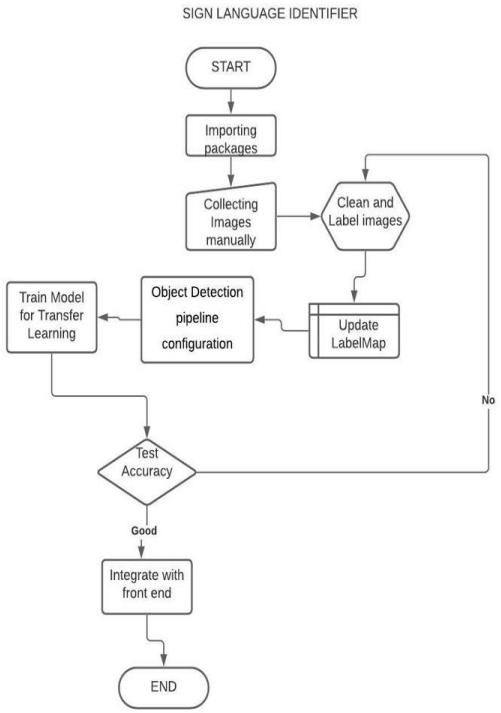


Fig.5 Implementation Workflow

5. PROPOSED SYSTEM

The proposed system is implemented or processed as follows:

1) Collect images for deep learning using webcam and OpenCV
2) Label images for sign language detection using Labelmg.
3) Setup Tensor flow Object Detection pipeline configuration
4) Use transfer learning to train a deep learning model
5) Detect sign language in real time using OpenCV

In this project, we have first collected images using webcam and OpenCV. After collecting the images, the second task is labelling those images for Sign Language Detection. We labeled the collected images using LabelImg. After labelling the images, it is mandatory to set up Tensorflow Object Detection pipeline configuration. Now, after setting up the Tensorflow Object Detection configuration, we will use Transfer Learning to train a deep learning model. Now, we are able to detect sign language in real time using OpenCV.

5.1. OVERVIEW

Throughout the long term, correspondence has played an indispensable job in return of data and sentiments in one's day to day existence. Sign language is the main medium through which deaf and mute individuals can interact with rest of the world through various hand motions. With the advances in machine learning, it is possible to detect sign language in real time. We have utilized the OpenCv python library, Tensorflow Object Detection pipeline and transfer learning to train a deep learning model that detects sign languages in Real time.

This section briefly presents overviews of the prior work and related searches carried out in the context of sign language conversion. In 2015, Taner Arsen published his research in the International Journal of Computer Science &Engineering Survey on converting sign language to text using a motion capture sensor. In 2019, Surejya Suresh published an IEEE paper titled Sign Language Recognition system using Deep Neural Networks. The main focus of this work is to create a vision based system, a Convolutional Neural Network (CNN) model, to identify six different sign languages from the images captured. G. A. Rao, K. Syamala, P.

V. V. Kishore and A. S. C. S. Sastry presented their research titled "Deep convolutional neural networks for sign language recognition" in the 2018 Conference on Signal Processing And Communication Engineering Systems.

A paper was published by C. J. L. Flores, A. E. G. Cutipa and R. L. Enciso, "Application of convolutional neural networks for static hand gestures recognition under different invariant features" in the 2017 IEEE XXIV International Conference on Electronics Electrical Engineering and Computing. All these cited works use signal sensing and processing techniques instead of image recognition using labels and machine learning models. Since it is a relatively budding and new area, not much work has been done in applying machine learning to develop a simplistic approach to leverage the user tagged parameters.

5.2. FEATURES

- User-friendly based GUI built using industrial standard PyQT5.
- Real time American standard character detection based on gesture made byuser.
- Customized gesture generation.
- Forming a stream of sentences based on the gesture made after a certaininterval of time.

5.3. TECHNOLOGY USED

TensorFlow: TensorFlow is an open-source artificial intelligence programme that uses data flow graphs to generate models. It allows programmers to create large-scale neural networks with multiple layers. TensorFlow is mostly used for classification, perception, comprehension, discovery, prediction, and creation. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019. TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

Object Detection API: It is an open source TensorFlow API to locate objects in an image and identify it. Object detection API is the framework for creating a deep learning network that solves object detection problems. There are already pretrained models in their framework

which they refer to as Model Zoo. This includes a collection of pretrained models trained on the COCO dataset, the KITTI dataset, and the Open Images Dataset. These models can be used for inference if we are interested in categories only in this dataset. They are also useful for initializing your models when training on the novel dataset.

OpenCV: OpenCV is an open-source, highly optimised Python library targeted at tackling computer vision issues. It is primarily focused on real-time applications that provide computational efficiency for managing massive volumes of data. It processes photos and movies to recognise items, people, and even human handwriting. It is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source Apache 2 License. Starting with 2011, OpenCV features GPU acceleration for real-time operations. OpenCV's application areas include:2D and 3D feature toolkits, Egomotion estimation, Facial recognition system, Gesture recognition, Human–computer interaction (HCI), Mobile robotics, Motion understanding, Object detection Segmentation and recognition, Stereopsis stereo vision: depth perception from 2 cameras, Structure from motion (SFM), Motion tracking Augmented reality etc. To support some of the above areas, OpenCV includes a statistical machine learning library that contains: Boosting, Decision tree learning, Gradient boosting trees, Expectation-maximization algorithm, k-nearest neighbor algorithm, Naive Bayes classifier, Artificial neural networks, Random forest, Support vector machine (SVM) & Deep neural networks (DNN).

LabelImg: LabelImg is a graphical image annotation tool that labels the bounding boxes of objects in pictures. LabelImg is a free, open source tool for graphically labeling images. It's written in Python and uses QT for its graphical interface. It's an easy, free way to label a few hundred images to try out your next object detection project.

SSD Mobilenet: The mobile-ssd models is a Single- Shot multibox detection network intended to perform detection. It works parallel with a convolutional neural network in Object Detection models. The SSD architecture is a single convolution network that learns to predict bounding box locations and classify these locations in one pass. Hence, SSD can be trained end-to-end. The SSD network consists of base architecture (MobileNet in this case) followed by several convolution layers:

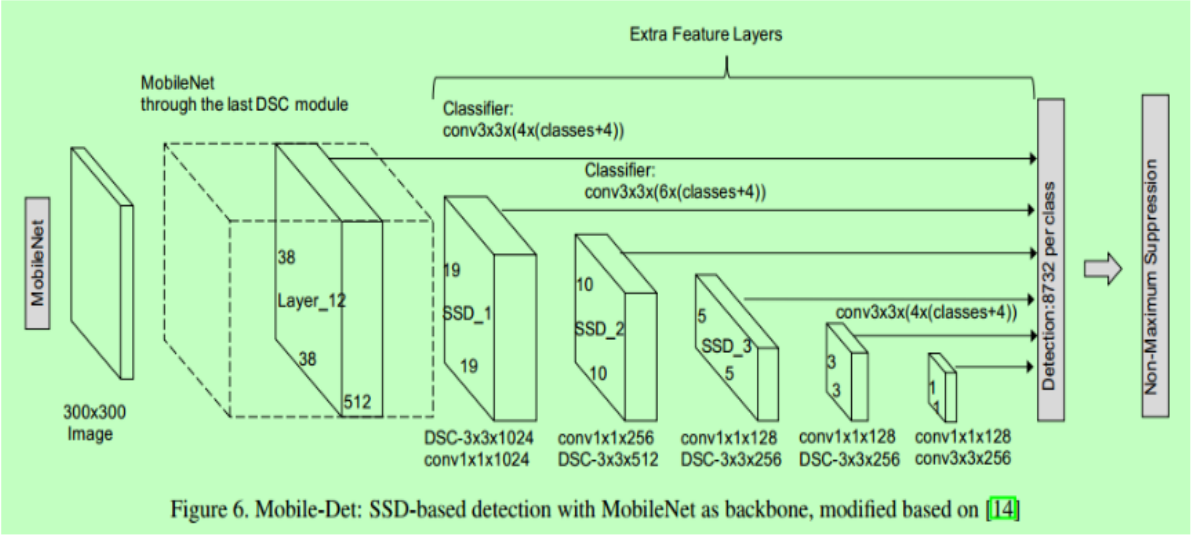


Fig. 6: SSD Mobilenet Layered Architecture

By using SSD, we only need to take one single shot to detect multiple objects within the image, while regional proposal network (RPN) based approaches such as R-CNN series that need two shots, one for generating region proposals, one for detecting the object of each proposal. Thus, SSD is much faster compared with two-shot RPN-based approaches.

Transfer Learning: Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. It is a popular approach where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems. In simpler words, it is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems. To put it simply—a model trained on one task is repurposed on a second, related task as an optimization that allows rapid progress when modelling the second task. By applying transfer learning to a new task, one can achieve significantly higher performance than training with only a small amount of data. Transfer learning is so common that it is rare to train a model for an image or natural language processing-related tasks from scratch. Instead, researchers and data scientists prefer to start from a pre-trained model that already knows how to classify

objects and has learned general features like edges, shapes in images. ImageNet, AlexNet, and Inception are typical examples of models that have the basis of Transfer learning.

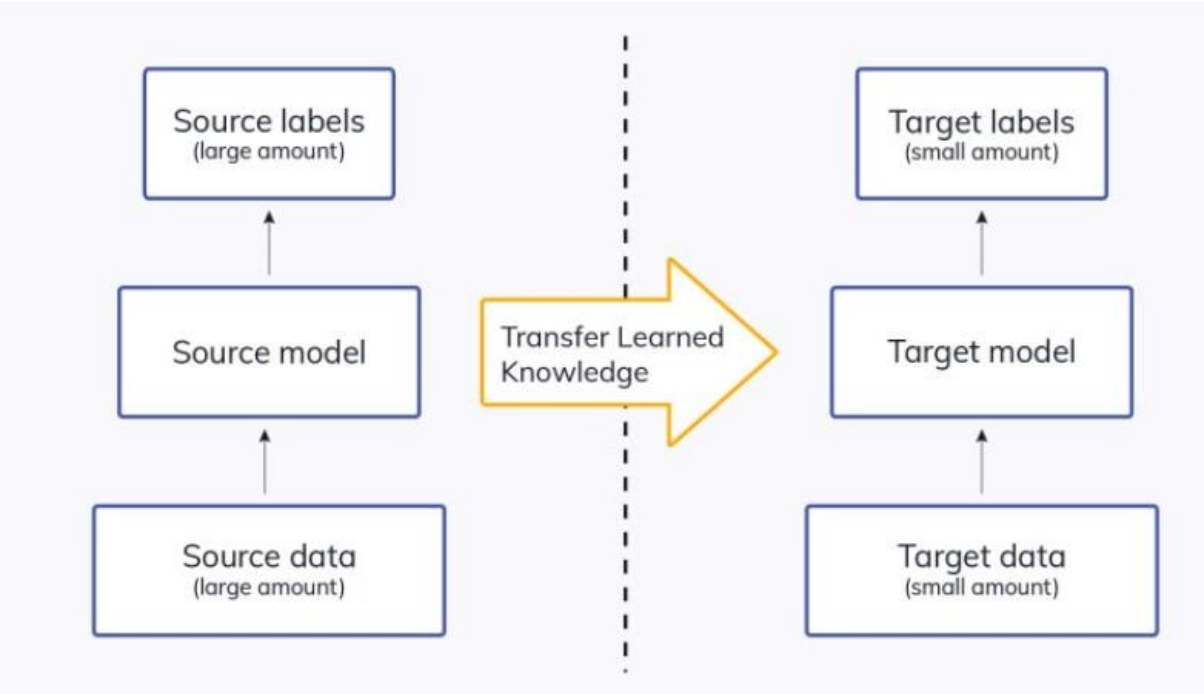


Fig 7: Transfer Learning

Gesture Recognition: Gesture recognition is a computing process that attempts to recognize and interpret human gestures through the use of mathematical algorithms. Gesture recognition technology that is vision based uses a camera and motion sensor to track user movements and translate them in real time. In simpler words, it is the mathematical interpretation of a human motion by a computing device. Gesture recognition, along with facial recognition, voice recognition, eye tracking and lip movement recognition are components of what developers refer to as a perceptual user interface (PUI). The goal of PUI is to enhance the efficiency and ease of use for the underlying logical design of a stored program, a design discipline known as usability. In personal computing, gestures are most often used for input commands. Recognizing gestures as input allows computers to be more accessible for the physically-impaired and makes interaction more natural in a gaming or 3-D virtual reality environment. Hand and body gestures can be amplified by a controller that contains accelerometers and gyroscopes to sense tilting, rotation and acceleration of movement -- or the computing device can be outfitted with a camera so that software in the device can recognize and interpret specific gestures. A wave of the hand, for instance, might terminate the program. In addition to the technical challenges of implementing gesture recognition, there are also social challenges.

Gestures must be simple, intuitive and universally acceptable. The study of gestures and other nonverbal types of communication is known as kinesics.

Convolutional Neural Networks: A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyse visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the downsampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

6. RESULTS AND DISCUSSION

The model was developed using a pre-trained model SSDmobile net v2 using a transfer learning technique. The practice of applying a model that has been trained on one problem to a second, similar problem in some way is known as transfer learning. Transfer learning is a deep learning technique that involves training a neural network model on a similar problem to the one being addressed before being applied to the present problem. After that, one or more layers from the learnt model are used to train a new model on the problem of interest. The Mobile Net SSD model is a single- shot multibox detection (SSD) network that identifies objects by scanning the pixels of an image that are inside the bounding box coordinates and class probabilities. In contrast to standard residual models, the model's design is built on the concept of inverted residual structure, in which the residual block's input and output are narrow bottleneck layers. Intermediate layer nonlinearities are also removed, and lightweight depth wise convolution is used. This model is included in the TensorFlow object detection API.

6.1. IMPLEMENTATION

6.1.1. Creating Dataset

In [1]:
```
import cv2
import os
import time
import uuid
```

In [2]:
```
IMAGES_PATH = 'Tensorflow/workspace/images/collectedimages'
```

In [3]:
```
labels = ['hello', 'thanks', 'yes', 'no', 'iloveyou']
number_imgs = 15
```

In [4]:
```
for label in labels:
    !mkdir {'Tensorflow\workspace\images\collectedimages\\'+label}
    cap = cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_imgs):
        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()
```

```
Collecting images for hello
Collecting images for thanks
Collecting images for yes
Collecting images for no
Collecting images for iloveyou
```

6.1.2. Training and Detection using tensorflow

0. Setup Paths

In [1]:
```
WORKSPACE_PATH = 'Tensorflow/workspace'
SCRIPTS_PATH = 'Tensorflow/scripts'
APIMODEL_PATH = 'Tensorflow/models'
ANNOTATION_PATH = WORKSPACE_PATH+'/annotations'
IMAGE_PATH = WORKSPACE_PATH+'/images'
MODEL_PATH = WORKSPACE_PATH+'/models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH+'/pre-trained-models'
CONFIG_PATH = MODEL_PATH+'/my_ssd_mobnet/pipeline.config'
CHECKPOINT_PATH = MODEL_PATH+'/my_ssd_mobnet/'
```

1. Create Label Map

In [2]:
```
labels = [{'name':'hello', 'id':1},
          {'name':'yes', 'id':2},
          {'name': 'no', 'id':3},
          {'name': 'Thank you', 'id': 4},
          {'name': 'i love you', 'id': 5},]

with open(ANNOTATION_PATH + '\label_map.pbtxt', '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')
```

2. Create TF records

In [3]:
```
!python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH + '/train'}
-l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/train.record'}
!python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x{IMAGE_PATH + '/test'}
-l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/test.record'}
```

```
Successfully created the TFRecord file: Tensorflow/workspace/annotations/train.record
Successfully created the TFRecord file: Tensorflow/workspace/annotations/test.record
```

3. Copy Model Config to Training Folder

In [4]:
```
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
```

In [5]:
```
!mkdir {'Tensorflow\workspace\models\\'+ CUSTOM_MODEL_NAME}
!copy {PRETRAINED_MODEL_PATH +'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config'}
{MODEL_PATH+'/'+CUSTOM_MODEL_NAME}
```

```
A subdirectory or file Tensorflow\workspace\models\my_ssd_mobnet already exists.
The syntax of the command is incorrect.
```

4. Update Config For Transfer Learning

In [6]:
```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

In [7]:
```
CONFIG_PATH = MODEL_PATH+'/'+CUSTOM_MODEL_NAME+'/pipeline.config'
```

In [8]:
```
config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
```

In [9]:
```
config
```

```
    num_classes: 5
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
```

In [10]:
```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

In [11]:
```
pipeline_config.model.ssd.num_classes = 5
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:]
= [ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:]
= [ANNOTATION_PATH + '/test.record']
```

In [12]:
```
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
    f.write(config_text)
```

5. Train the model

In [13]:
```
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{}
--pipeline_config_path={}/{}/pipeline.config --num_train_steps=5000""".format(
    APIMODEL_PATH, MODEL_PATH,CUSTOM_MODEL_NAME,MODEL_PATH,CUSTOM_MODEL_NAME))
```

```
python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspace/models
y_ssd_mobnet/pipeline.config --num_train_steps=5000
```

6. Load Train Model From Checkpoint

In [6]:
```
import os
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
```

In [7]:
```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-6')).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
```

7. Detect in Real-Time

In [8]:
```
import cv2
import numpy as np
```

In [9]:
```
category_index = label_map_util.create_category_index_from_labelmap(
    ANNOTATION_PATH+'/label_map.pbtxt')
```

In [10]:
```
# Setup capture
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

In [11]:
```
while True:
    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=.5,
                agnostic_mode=False)

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

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

In [12]:

```
detections = detect_fn(input_tensor)
```

6.1.3. Convert Model to Tensorflow Js Model

9. Save Graph

In [7]:

```
print("""python {}/research/object_detection/exporter_main_v2.py --input_type=image_tensor --pipeline_config_path={}/{}/pipel
```

```
python Tensorflow/models/research/object_detection/exporter_main_v2.py --input_type=image_tensor --pipeline_config_path=Tens
orflow/workspace/models/my_ssd_mobnet/pipeline.config --trained_checkpoint_dir=Tensorflow/workspace/models/my_ssd_mobnet/ --
output_directory=Tensorflow/workspace/models/my_ssd_mobnet/export
```

10. Convert to TFJS model

In [8]:

```
!pip install tensorflowjs
```

```
...
```

In [11]:

```
"""tensorflowjs_converter --input_format=tf_saved_model --output_node_names='detection_boxes,detection_classes,detection_feat
```

Out[11]:

```
"tensorflowjs_converter --input_format=tf_saved_model --output_node_names='detection_boxes,detection_classes,detection_featu
res,detection_multiclass_scores,detection_scores,num_detections,raw_detection_boxes,raw_detection_scores' --output_format=tf
js_graph_model --signature_name=serving_default Tensorflow/workspace/models/my_ssd_mobnet/export/saved_model Tensorflow/work
space/models/my_ssd_mobnet/converted"
```

In [ ]:

6.1.4. utilities.js :- Creating labelmap and Drawing function

```
// Define our labelmap
const labelMap = {
    1:{name:'Hello', color:'red'},
    2:{name:'Thank You', color:'yellow'},
    3:{name:'I Love You', color:'lime'},
    4:{name:'Yes', color:'blue'},
    5:{name:'No', color:'purple'},
}

// Define a drawing function
export const drawRect = (boxes, classes, scores, threshold, imgWidth,
imgHeight, ctx)=>{
    for(let i=0; i<=boxes.length; i++){
        if(boxes[i] && classes[i] && scores[i]>threshold){
            // Extract variables
            const [y,x,height,width] = boxes[i]
```

```
            const text = classes[i]

            // Set styling
            ctx.strokeStyle = labelMap[text]['color']
            ctx.lineWidth = 10
            ctx.fillStyle = 'white'
            ctx.font = '30px Arial'

            // DRAW!!
            ctx.beginPath()
            ctx.fillText(labelMap[text]['name'] + ' - ' +
Math.round(scores[i]*100)/100, x*imgWidth, y*imgHeight-10)
            ctx.rect(x*imgWidth, y*imgHeight, width*imgWidth/2,
height*imgHeight/1.5);
            ctx.stroke()
        }
    }
}
```

6.1.5. App.js :- Drawing canvas using request animation frame

```
// Import dependencies
import React, { useRef, useState, useEffect } from "react";
import * as tf from "@tensorflow/tfjs";
import Webcam from "react-webcam";
import "./App.css";
import { nextFrame } from "@tensorflow/tfjs";
// 2. TODO - Import drawing utility here
// e.g. import { drawRect } from "./utilities";
import {drawRect} from "./utilities";

function App() {
  const webcamRef = useRef(null);
  const canvasRef = useRef(null);

  // Main function
  const runCoco = async () => {
    // 3. TODO - Load network
    // e.g. const net = await cocossd.load();
    // https://realtimesignlanguagedetection.s3.us-east.cloud-object-
storage.appdomain.cloud/model.json
    const net = await
tf.loadGraphModel('https://realtimesignlanguagedetection.s3.us-east.cloud-
object-storage.appdomain.cloud/model.json')

    //  Loop and detect hands
    setInterval(() => {
      detect(net);
    }, 16.7);
  };

  const detect = async (net) => {
    // Check data is available
    if (
      typeof webcamRef.current !== "undefined" &&
      webcamRef.current !== null &&
      webcamRef.current.video.readyState === 4
```

```
    ) {
      // Get Video Properties
      const video = webcamRef.current.video;
      const videoWidth = webcamRef.current.video.videoWidth;
      const videoHeight = webcamRef.current.video.videoHeight;

      // Set video width
      webcamRef.current.video.width = videoWidth;
      webcamRef.current.video.height = videoHeight;

      // Set canvas height and width
      canvasRef.current.width = videoWidth;
      canvasRef.current.height = videoHeight;

      // 4. TODO - Make Detections
      const img = tf.browser.fromPixels(video)
      const resized = tf.image.resizeBilinear(img, [640,480])
      const casted = resized.cast('int32')
      const expanded = casted.expandDims(0)
      const obj = await net.executeAsync(expanded)
      console.log(obj)

      const boxes = await obj[1].array()
      const classes = await obj[2].array()
      const scores = await obj[4].array()

      // Draw mesh
      const ctx = canvasRef.current.getContext("2d");

      // 5. TODO - Update drawing utility
      // drawSomething(obj, ctx)
      requestAnimationFrame(()=>{drawRect(boxes[0], classes[0], scores[0],
0.8, videoWidth, videoHeight, ctx)});

      tf.dispose(img)
      tf.dispose(resized)
      tf.dispose(casted)
      tf.dispose(expanded)
      tf.dispose(obj)

    }
  };

  useEffect(()=>{runCoco()},[]);

  return (
    <div className="App">
      <header className="App-header">
        <Webcam
          ref={webcamRef}
          muted={true}
          style={{
            position: "absolute",
            marginLeft: "auto",
            marginRight: "auto",
            left: 0,
```

```
            right: 0,
            textAlign: "center",
            zindex: 9,
            width: 640,
            height: 480,
          }}
        />

        <canvas
          ref={canvasRef}
          style={{
            position: "absolute",
            marginLeft: "auto",
            marginRight: "auto",
            left: 0,
            right: 0,
            textAlign: "center",
            zindex: 8,
            width: 640,
            height: 480,
          }}
        />
      </header>
    </div>
  );
}

export default App;
```

6.2. RESULT

Fig.8: Live detection of signs

The detection box is dynamic and expands its dimensions based on the size of the input image processed. AP (Average precision) is a popular metric in measuring the accuracy of object detectors like the SSD model. Average precision computes the average precision value for recall value over 0 to 1.

6.3. EVALUATION OF IMPLEMENTED WORK

The usage of deep learning model is proven to be more efficient than machine learning algorithms in terms of the computation speed and power of the model.

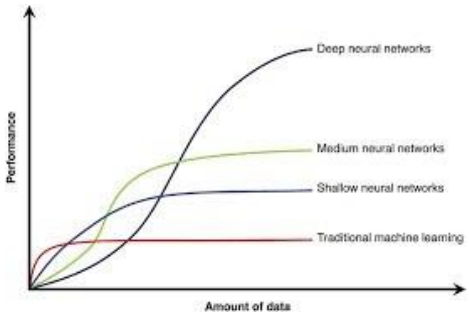


Fig.9 :Average precision and recall values

The training and validation accuracy of the deep learning model is plotted. An 80% training accuracy and average of 60% validation accuracy is obtained as represented in Fig.8.

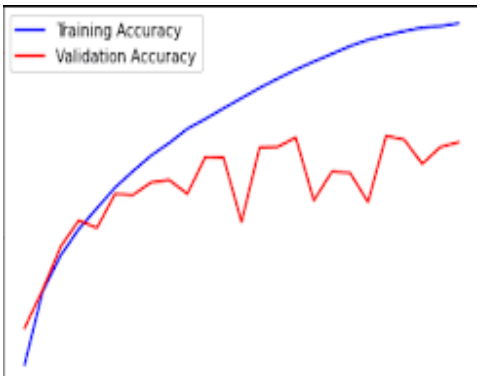


Fig.10: Training and Validation Accuracy

The model is successfully trained and run with nocompilation errors. An average accuracy of 60-80% is achieved in the detection.

The detection occurs within a proximity of 40-50 metrefrom the system camera. The detection box is dynamic and expands its dimensions based on the size of the input image processed. AP (Average precision) is a popular metric in measuring the accuracy of object detectors like

the SSD model. Average precision computes the average precision value for recall value over 0 to 1.

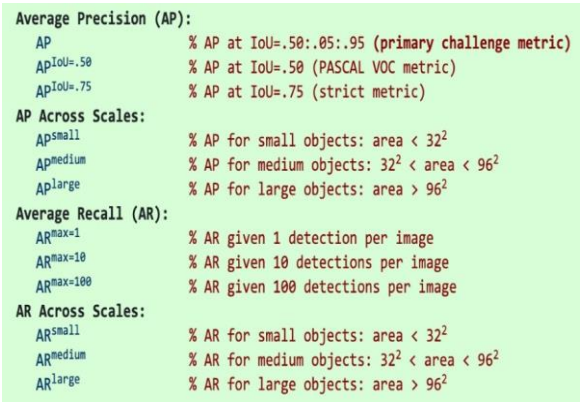


Fig.11: Average precision and recall values

A second run through was carried out to increase the accuracy by feeding more images as input and re- training the model. The accuracy improved by a marginof 5-10% indicating that with more number of samples the accuracy of the model is bound to inrease.

7. CONCLUSION

The fundamental goal of a sign language detecting system is to provide a practical mechanism for normal and deaf individuals to communicate through hand gestures. The suggested method may be used with a webcam or anyother in-built camera that detects and processes indicators for recognition. We may deduce from the model's findings that the suggested system produces reliable results under conditions of regulated light and intensity. Furthermore, new motions may be simply incorporated, and more photographs captured from various angles and frames willsupply the model with greater accuracy. As a result, by expanding the dataset, the model may simply be scaled up to a vast size. Environmental issues such as low light intensity and an unmanaged backdrop are some of the model's limitations cause decrease in the accuracy of the detection. Therefore, we'll work next to overcome these flaws and also increase the dataset for more accurate results.

From this project/application we have tried to overshadow some of the major problems faced by the disabled persons in terms of talking. We found out the root cause of why they can't express more freely. The result that we got was the other side of the audienceare not able to interpret what these persons are trying to say or what is the message that they want to convey. Thereby this application serves the person who wants to learn and talk in sign languages.With this application a person will quickly adapt various gestures and their meaning as per ASL standards. They can quickly learn what alphabet is assigned to which gesture. Add-on to this custom gesture facility is also provided along with sentence formation. Auser need not be a literate person if they know the action of the gesture, they can quicklyform the gesture and appropriate assigned character will be shown onto the screen.

Concerning to the implementation, we have used TensorFlow framework, with keras API. And for the user feasibility complete front-end is designed using PyQT5. Appropriate user-friendly messages are prompted as per the user actions along with whatgesture means which character window. Additionally, an export to file module is also provided with TTS(Text-To-Speech) assistance meaning whatever the sentence was formed a user will be able to listen to it and then quickly export along with observing what gesture he/she made during the sentence formation.

7.1. FUTURE SCOPE, IMPROVEMENTS AND APPLICATION

7.1.1. FUTURE SCOPES & ENHACEMENTS

- It can be integrated with various search engines and texting application suchas google, WhatsApp. So that even the illiterate people could be able to chatwith other persons, or query something from web just with the help of gesture.
- This project is working on image currently, further development can lead to detecting the motion of video sequence and assigning it to a meaningful sentence with TTS assistance.
- Since the project is done on a low scale with fewresources and limited capabilities, there are many future enhancements that can be done to make it a real world application for use. A GPU machine can be used to achieve more accuracy and trained with more sign images. The model can be fed and used with an external camera for capturing and detecting sign languages from a longer distance.
- We can develop a model for ISL word and sentence level recognition. This will require a system that can detect changes with respect to the temporal space.
- The model can also be programmed with teleconferencing software like Google Meet, Zoom as a functionality extension to enable deaf and dumb people to interact in the virtual medium using the sign detection model.
- We can develop a complete product that will help the speech and hearing impaired people, and thereby reduce the communication gap.
- The adaptation of our concept to other sign languages, such as Indian and American sign languages. To improve the neural network's ability to recognise symbols, it will be further trained. Enhancement of the model's ability to recognise facial emotions.

7.1.2. APPLICATION

The collection can easily be enlarged and updated to match the user's needs, and it might be a big step in bridging the communication gap between the deaf and the dumb. Meetings held on a worldwide scale can become simpler to grasp for disabled people, and the worth of their hard work can be communicated utilising the sign detection technique. The concept may be used by anybody with a little grasp of technology, making it accessible to all. This technique might be implemented in elementary schools to teach children sign language at a young age.

8. REFERENCES

- https://www.researchgate.net/publication/ 282839736_Sign_Language_Converter
- http://sersc.org/journals/index.php/IJAST/article/view/ 20937/10563
- https://tensorflow-object-detection-api- tutorial.readthedocs.io/en/latest/
- https://academic.oup.com/jdsde/article/11/4/421/411839
- https://link.springer.com/chapter/10.1007/978-3-642- 02707-9_3
- https://www.sciencedirect.com/science/article/pii/ S1877050918321331
- https://www.ijert.org/sign-language-to-text-and-speech-translation-in-real-time-using-convolutional-neural- network
- https://www.ripublication.com/ijaer18/ ijaerv13n9_90.pdf
- https://www.researchgate.net/figure/Comparison- accuracy-Faster-R-CNN-R-FCN-SSD-and-YOLO-models
- http://reports.ias.ac.in/report/19049/real-time-indian- sign-language-recognition