APPLICATION CONTROL USING HAND GESTURE INTERPRETATION BASED ON CNN

A project dissertation submitted to Bharathidasan University
In partial fulfillment of the requirements
For the award of the Degree of

MASTER OF SCIENCE IN DATA SCIENCE

Submitted by

VIVEKA.B
(205229132)

Guided by

Dr. B. Karthikeyan., M.S (IT)., M.Phil., Ph.D.
Assistant Professor

DEPARTMENT OF DATA SCIENCE
BISHOP HEBER COLLEGE (AUTONOMOUS)
(Nationally Reaccredited at the 'A' Grade by NAAC with the CGPA of 3.58 out of 4)
(Recognized by UGC as "College with Potential for Excellence")
(Affiliated to Bharathidasan University)

TIRUCHIRAPPALLI 620017

MAY 2022

DECLARATION

I hereby declare that the project work presented is originally done by me under the guidance of **Dr. Karthikeyan M.S (IT), M.Phil., Ph.D. Department of Data Science, Bishop Heber College (Autonomous), Tiruchirappalli 620017**, and has not been included in any other thesis/project submitted for any other degree.

Name of the Candidate	**:**	**B.VIVEKA**
Register Number	**:**	**205229132**
Batch	**:**	**2020-2022**

Signature of the Candidate

DEPARTMENT OF DATA SCIENCE BISHOP HEBER COLLEGE (AUTONOMOUS)

(Nationally Reaccredited at the 'A' Grade by NAAC with the CGPA of 3.58 out of 4)

(Recognized by UGC as "College with Potential for Excellence")

(Affiliated to Bharathidasan University)

TIRUCHIRAPPALLI 620017

Date:

Course Title: Project **Course Code: P20DS4PJ**

BONAFIDE CERTIFICATE

This is to certify that the project work titled **"APPLICATION CONTROL USING HAND GESTURE INTERPRETATION BASED ON CNN"** is a bonafide record of the project work done by **B viveka, (205229132),** in partial fulfilment of the requirements for the award of the degree of **MASTER OF SCIENCE IN DATA SCIENCE** during the period **2020 - 2022**

The Viva-Voce examination for the candidate **B. VIVEKA, 205229132** was held on________________________________.

Signature of the HOD **Signature of the guide**

1.

2.

ACKNOWLEDGEMENTS

I take this opportunity to express my thanks to **Dr. D. PAUL DHAYABARAN., M.Sc., PGDCA, M.Phil., Ph.D., Principal** Bishop Heber College, Trichy.

I like to express my gratitude and deep thanks to **Dr. K. RAJKUMAR M.Sc., M.Phil., Ph.D.,Head of the Department of Data Science** and who has been a source of encouragement and moral strength throughout our study period.

This project would not be possible without the motivation and guidance of our internal guide **Dr. B. KARTHIKEYAN., M.S (IT)., M.Phil., Ph.D.** he has been the backbone of my project, he has been encouraging and been a moral support to finish this project

I am extremely thankful to my family members and my dear friends who helped a lot in the completion of the project.

B VIVEKA

Curiginal

Document Information

Analyzed document	205229132.pdf (D138175939)
Submitted	2022-05-27T10:12:00.0000000
Submitted by	Librarian
Submitter email	librarypds@bhc.edu.in
Similarity	15%
Analysis address	librarypds.bhc@analysis.urkund.com

Sources included in the report

W	URL: https://www.sciencedirect.com/science/article/pii/S2352340920305709 Fetched: 2020 08-06T18:11:26.4830000	1
W	URL: https://www.researchsquare.com/article/rs-1495878/latest.pdf Fetched: 2022 05-27T10:12:20.7730000	3
W	URL: https://www.cluzters.ai/Model/2597/detection-of-tuberculosis-disease-using-tensorflow-technique Fetched: 2022-05-27T10:12:19.5000000	1
SA	**Final_report_10544512.pdf** Document Final_report_10544512.pdf (D106263909)	4
SA	**17C134_Report Final Submission.docx** Document 17C134_Report Final Submission.docx (D104574020)	3

ABSTRACT

Hand gesture recognition provides an intelligent and natural way of human computer interaction (HCI). Its applications range from medical rehabilitation to consumer electronics control (e.g. mobile phone). In order to distinguish hand gestures, various kinds of sensing techniques are utilized to obtain signals for pattern recognition. Gestures a non-verbal form of communication provides the HCI interface. The goal of gesture recognition is to create a system which can identify specific human gestures and use them to convey information or for device control. Real-time vision-based hand gesture recognition is considered to be more and more feasible for HCI with the help of latest advances in the field of computer vision and pattern recognition.

This project deals with a discussion of various techniques, methods and algorithms related to gesture recognition. The hand gesture is the easiest and most natural way of communication. Hand gesture recognition has the various advantages of being able to communicate with the Technology through basic sign language. The gesture will be able to reduce the use of the most prominent hardware devices which areused to control the activities of a computer. The system has two major advantages. First, it is highly modularized, and each of the three steps is capsuled from others; second, the edge/contour detection of hand, as well as gesture recognition, is an add-on layer, which can be easily transplanted to other applications.

Edge detection is one of the most commonly used operations in image analysis, and there are probably more algorithms in the literature for enhancing and detecting edges than any other single subject. The reason for this is that edges form the outline of an object. An edge is the boundary between an object and the background and indicates the boundary between overlapping objects. This means that if the edges in an image can be identified accurately, all of the objects can be located and basic properties such as area, perimeter, and shape can be measured.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Motivation

In computer vision and machine learning, hand gesture detection for human-computer interaction is a hot topic. Hand segmentation and feature extraction are critical steps in a vision-based hand gesture recognition system for hand posture classification. One of the main goals of hand recognition research is to develop systems that can recognise specific gestures and use them to convey data and control a device. Feature extraction methods are used to extract information from photos that aids in the gesture recognition process. The pre-processing stage prepares the input image and extracts features that will be used by classification algorithms later on. All commercialtranslation services are now human-based, which makes them costly due to the required experience. Sign Language Recognition aims to create algorithms and methods for correctly identifying and comprehending a sequence of produced signs. Many approaches to SLR to treat the problem correctly as Gesture Recognition (GR). So far, researchhas focused on identification of optimum characteristics and classification methods to accurately label a given sign from a number of potential signs.

1.2 Existing Systems and Solutions

There are a variety of systems for commanding a robot with gestures. Adaptive colour segmentation, hand locating and labelling with blocking, morphological filtering, and subsequently template matching and skeletonizing are used in several gesture recognition systems. Due to template matching, this does not allow dynamicity for gesture inputs. Some systems use a machine interface device to give the robot real-time gestures. Many input methods need physical touch, but there are also several that enable input to the application without requiring physical touch, relying instead on other human qualities such as speech or hand gestures. There are two techniques to gesture recognition. Vision-based a) and glove-based b). Sensors or gloves are used to identify hand gestures in a glove-based technique. In a glove-based technique, flex sensors, accelerometers, and other sensors are used. Static or dynamic gestures are possible. Hand postures are used in static gestures, and the image is taken with cameras. The collected images are sent to be analysed, which is done via segmentation. The skin detection technique extracts the skin region from the input image, which may contain a variety of other objects in addition to the hand region. The image acquired using a.webcam is in the RGB colour model in image acquisition. Because the regions that relate to the skin may be easily distinguished in the HSI model, this image was transformed into the HSI color model.

1.2.1 Disadvantages:

- The system in which work is being done must have hardware control to detect the user's hands.
- If there's a lot going on in the background. When the lighting is changing, segmentation might be a challenging operation.
- Hand tracking has a lower segmentation accuracy.

1.3 Product Needs and Proposed System

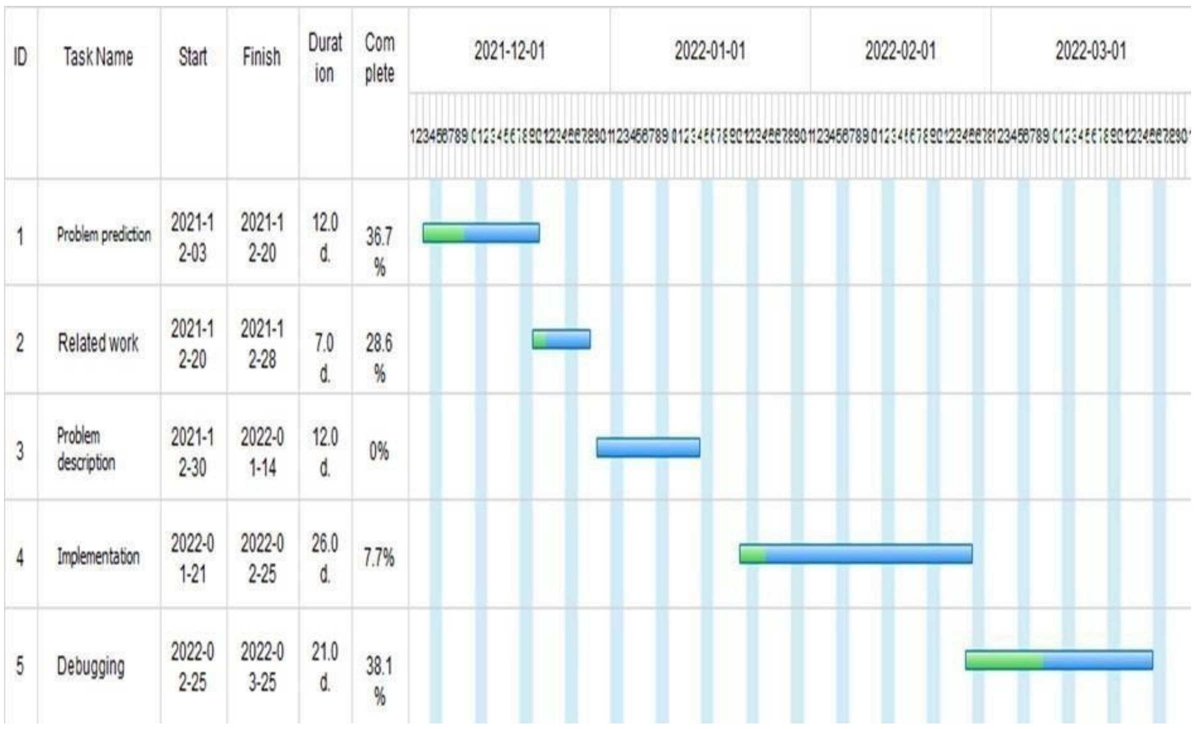
The robot is controlled by the user from a control station, which can be a laptop or a PC with a high-quality inbuilt or external webcam. This webcam is used to capture a live video stream of hand movements in order to produce robot orders. Gesture directives are given with the palm of the hand. There are five different types of gestures that are employed, each of which is explored further. The robot may move in any direction in the surroundings by applying five different sorts of orders to access the five programmers. Image Processing is used to process an image frame as an input. The gesture command is then extracted from the processed image. One of the five possible commands for this gesture command is given. A signal is generated from this gesture command and sent to the robot via the web camera.

The input image comes from a webcam. By removing the hue and saturation information from an RGB image, it is turned to a grayscale image. The resultant image's contrast is modified via the illumination control. Hand landmark detection is also used to detect finger landmarks. The Frame separation method builds structural elements by employing a disc form. The landmark points can be detected using a hand land mark model. The output generates finger count landmark values. Convolutional neural networks are used to classify and run different applications based on finger counts.

1.3.1. Advantages

- Everyone wants to live a convenient life, as we all know.
- Simple, fast, and easy to implement
- Speed and sufficient reliability for the recognition system

1.4 GRANT CHART



ID	Task Name	Start	Finish	Duration	Complete
1	Problem prediction	2021-12-03	2021-12-20	12.0 d.	36.7 %
2	Related work	2021-12-20	2021-12-28	7.0 d.	28.6 %
3	Problem description	2021-12-30	2022-01-14	12.0 d.	0%
4	Implementation	2022-01-21	2022-02-25	26.0 d.	7.7%
5	Debugging	2022-02-25	2022-03-25	21.0 d.	38.1 %

Figure 1: Grant Chart

CHAPTER 2
RELATED WORK AND SOLUTIONS REVIEW

2.1 Dynamic Hand Gesture Recognition System Using OpenCV

Basically, this paper used basic techniques like gesture representation, motion capture, smoothing, and generating gestures. In the proposed system, there is no need to specify the start and end positions of gestures because the machine returns to its initial state automatically in the event of an incorrect gesture input or a timeout and the accuracy is almost 73%, Kılıboz, *et. al*[1]

In the paper, Vision-based and sensor-based techniques are the two types of gesture recognition techniques. This study proposes a technique for database-driven hand gesture recognition based on a skin color model approach and thresholding approach, as well as an effective template matching using PCA, for vision-based hand gesture recognition. And it shows 91.25% average accuracy and 0.098251 seconds average recognition time. And in the paper is a hand gesture recognition algorithm proposed here by Shrivastava, *et.al* [3]

Detection and tracking, feature extraction, and training and recognition are the three stages of the system. The Baum-Welch algorithm is used for training, with the Left-Right Banded (LRB) topology, and the Forward algorithm is used for recognition, with an average recognition rate of over 90% for solitary hand movements, Wang, Guojiang, *et.al* [4]

2.2 Hand Gesture Recognition In-Depth Data Using Dynamic Time Warping

The paper proposes a method for the simultaneous localization and recognition of dynamic hand gestures is proposed. At the core of this method is a dynamic space-time warping (DSTW) algorithm that aligns a pair of query and model gestures in both space and time. Next, a state-based technique for the representation and recognition of gestures is presented [5] Bagri, *et.al*

The approach is illustrated by application to a range of gesture-related sensory data: the two- dimensional movements of a mouse input device, the movement of the hand-measured by a magnetic spatial position and orientation sensor, and, lastly, the changing eigenvector projection coefficients computed from an image sequence. The paper, the use of hand gestures provides an attractive alternative to cumbersome interface devices for human-computer interaction (HCI). In particular, visual interpretation of hand gestures can help in achieving the ease and naturalness desired for HCI. This hasmotivated a very active area concerned with computer vision-based analysis and interpretation of hand gestures Yun, *et.al* [6]

This paper presents algorithms and a prototype system for hand tracking and hand research posture recognition. Hand postures are represented in terms of hierarchies of multi-scale color image features at different scales, with qualitative inter-relations in terms of scale, position and orientation. In each image, detection of multi-scale color features is performed, Plouffe, *at.el* [7]

2.3 Multi-scale Deep Learning For Gesture Detection And Localization

This paper develops a depth-aware attention module (DAM) to exploit spatial relations in the depth feature Maps, refining the RGB and depth feature maps across a bottleneck structure. The module establishes a lateral connection of the RGB and depth paths and provides a depth-aware salient map to both paths. The experimental results demonstrated that the proposed network improved the accuracy (+0.83%) and F score (+1.55%) [8] Chai, Xiujuan *et.al* .

This paper focuses on building a robust hand gesture recognition system using the Kinect sensor. To handle the noisy hand shape obtained from the Kinect sensor, we propose a novel distance metric for hand dissimilarity measure, called Finger-Earth Mover's Distance (FEMD) [9] Simão, *et.al.*

Hand-shapes corresponding to letters of the alphabet are characterized using appearance and depth images and classified using random forests. We compare classification using appearance and depth images and show a combination of both lead to best results, and validate on a dataset of four different users [10] Neverova, *et al.*

CHAPTER 3
DATA COLLECTION

3.1 Description of the Data

Gesture recognition technology is rapidly growing in the recent years due to the demands of many application such as computer game and sport, human robot interaction, assistant systems, sign language interpretation and e-commerce. One of the most important of gesture recognition is hand-gesture recognition.

The most efficient and expressive way of human communication is through hand gesture, which is a universally accepted language. It is pretty much expressive such that the dumb and deaf people could understand it. In this work, real-time hand gesture system is proposed. Experimental setup of the system uses fixed position low cost web camera high definition recording feature mounted on the top of monitor of computer or a fixed camera on a laptop, which captures snapshot using Red Green Blue [RGB] color space from fixed distance.

3.2 Source and Methods of Collecting Data

The runtime operations are managed by the webcam of the connected laptop or desktop. To capture a video, we need to create a Video Capture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Since we only use a single camera we pass it as '0'.

After that, can capture frame-by- frame of real time person hand. The infinite loop is used so that the web camera captures the frames in every instance and is open during the entire course of the program. We capture the live feed stream, frame by frame. Then we process each captured frame which is in RGB (default) color space.

CHAPTER 4
PREPROCESSING AND FEATURE SELECTION

4.1 Overview of Pre-processing Methods

4.1.1 Landmark Detection:

A motion is an intended movement of the full human body or its portions that is intended to convey a specific message. The signals are frequently expressed through hand positions, face expressions, or arm motions, and they can be static or dynamic. Gesture detection from digital photos and videos is a hot topic in computer vision that has gotten a lot of attention in recent years. Hand gesture identification necessitates the completion of a number of difficult computer vision and pattern recognition tasks, such as I human skin segmentation to identify hand regions from colour images, (ii) hand position estimation, (iii) hand tracking, and (iv) hand motion analysis and recognition.

Following the recognition of palms over the whole image, our subsequent hand landmark model uses regression to perform exact key point localization of 21 hand-knuckle coordinates within the observed hand regions. Even with partially visible hands and self-occlusions, the model develops a consistent internal hand posture representation. We manually labelled real-world photos with 21 coordinates to generate ground truth data. However, in our project, we only needed 12 co-ordinates, such as
Index finger mcp, Index finger PIP, Index finger DIP, Index finger TIP, Middle finger mcp, Middle finger PIP, Middle finger DIP, Middle finger TIP, Ring finger mcp, Ring finger PIP, Ring finger DIP, Ring finger TIP, Index finger DIP, Index finger TIP

Static image mode:

This argument's valid value is a Boolean value, which can be either True or False. The default condition is False, which is still valid for video streaming. For I say video streaming, I mean that it reduces processing latency by focusing on specific hands and localising them until it loses track of them. This can be useful when detecting hands in live streams or videos, but we need to detect landmarks on images, so we'll set the value to True.

Max num hands:

This parameter specifies the maximum number of hands that the model can detect in a single instance. By default, the value is 2, which makes sense because we'll want at least a pair of hands to be identified, but we can certainly adjust that.

Min detection confidence:

This parameter gives us the option to choose how rigidly we want our detection model to be, and in this case, it gives us the degree of confidence's threshold value. The optimal range for minimal detection confidence is [0.0,1.0], however it is set to 0.5 by default, which implies that if the confidence level drops below 50%, the hands will not be identified in the output image at all.

4.2 Overview of Feature Selection Methods

The maximum number of hands that the model can detect in a single instance is specified by this parameter. The value is set to 2 by default, which makes sense because we'll need at least two hands to be detected, but we can easily change it. The degree of confidence threshold value is determined by the minimum detection confidence parameter, which allows us to pick how strictly we want our detection model to be. The recommended range for lowest detection confidence is [0.0,1.0], however it is defaulted to 0.5, which means that if the confidence level drops below 50%, the hands will not be recognised in the output image at all.

.

4.3 Preprocessing and Feature Selection Steps

The approach of using region profiles has a number of flaws. For a specific object class, the (r,) plot will be multi-valued. As a result, the matching process becomes partially 2-D, resulting in complication and wasteful computing. As a result, we recommend the updated centroid profile. To begin, we obtain the contour for the provided hand region by utilising chain code. We also compute the centroid profile for each contour pixel. That is, we calculate the distance between the hand region's centroid and the contour boundary.

CHAPTER 5
MODEL DEVELOPMENT

5.1 Model Architecture

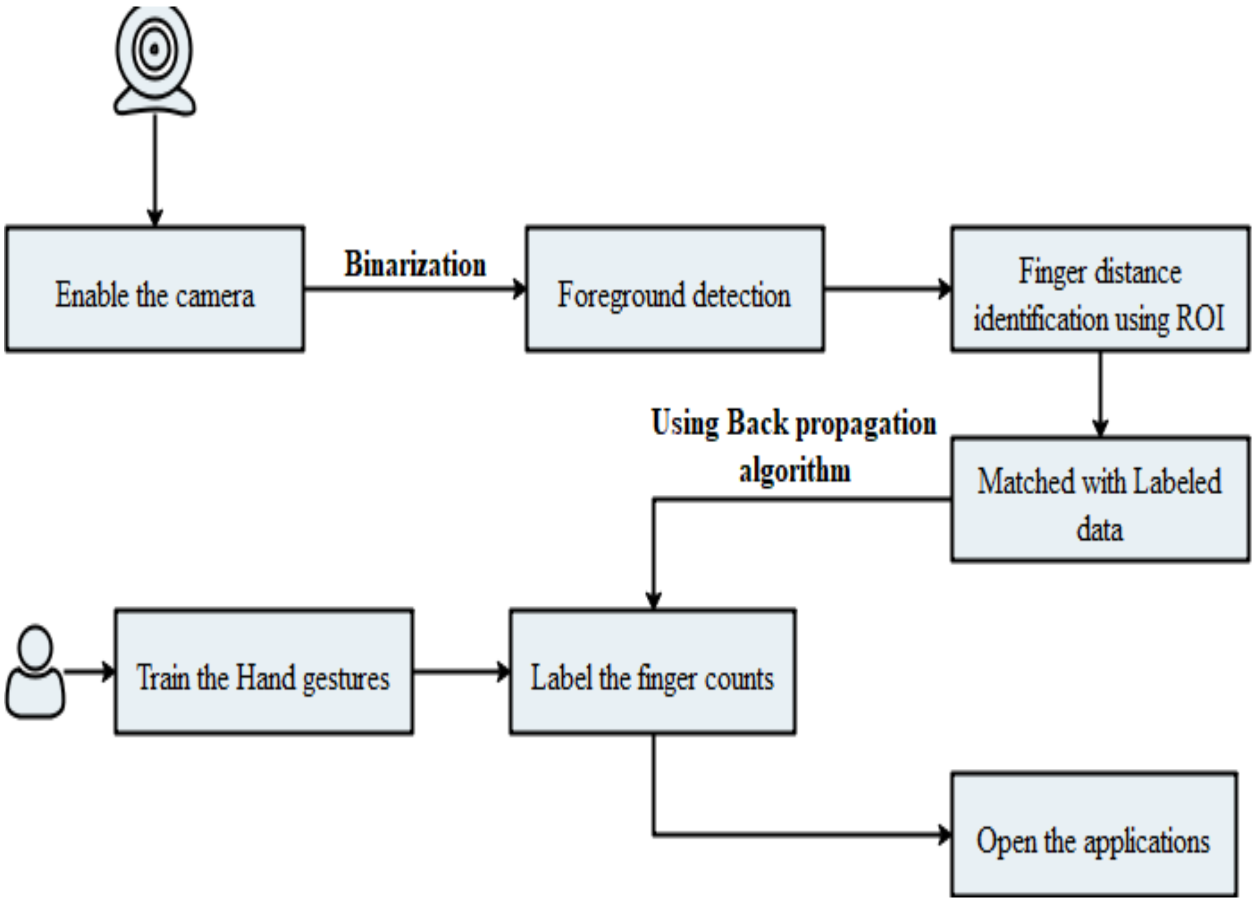


Figure 2: Model Architecture

5.2 Algorithms Applied

5.2.1 FINGER CLASSIFICATION: Mobi_net CNN

For Image Classification and Mobile Vision, MobileNet is a CNN architecture model. Other models exist, but what makes MobileNet unique is that it requires relatively little computational power to execute or apply transfer learning. This makes it ideal for mobile devices, embedded systems, and PCs with limited computing efficiency or no

GPU, without affecting the accuracy of the results greatly. It's also ideal for web browsers, which have limitations in terms of compute, graphics processing, and storage.
MobileNets, which are based on a streamlined design that leverages depthwise separable convolutions to generate light weight deep neural networks, are presented for mobile and embedded vision applications. Two simple global hyper-parameters are proposed to efficiently trade between latency and accuracy.
Depthwise separable filters, also known as Depthwise Separable Convolution, are the foundation of MobileNet. Another component that improves performance is the network structure. Finally, the breadth and resolution of the image can be adjusted to balance latency and accuracy.
Depthwise separable convolutions are a type of factorised convolution in which a standard convolution is factorised into a depthwise convolution and a 1111 convolution known as a pointwise convolution. The depthwise convolution in MobileNet applies a single filter to each input channel. After that, the pointwise convolution uses a 1111 convolution to combine the depthwise convolution's outputs. The difference between normal convolution and depthwise separable convolution is illustrated in the diagram below.

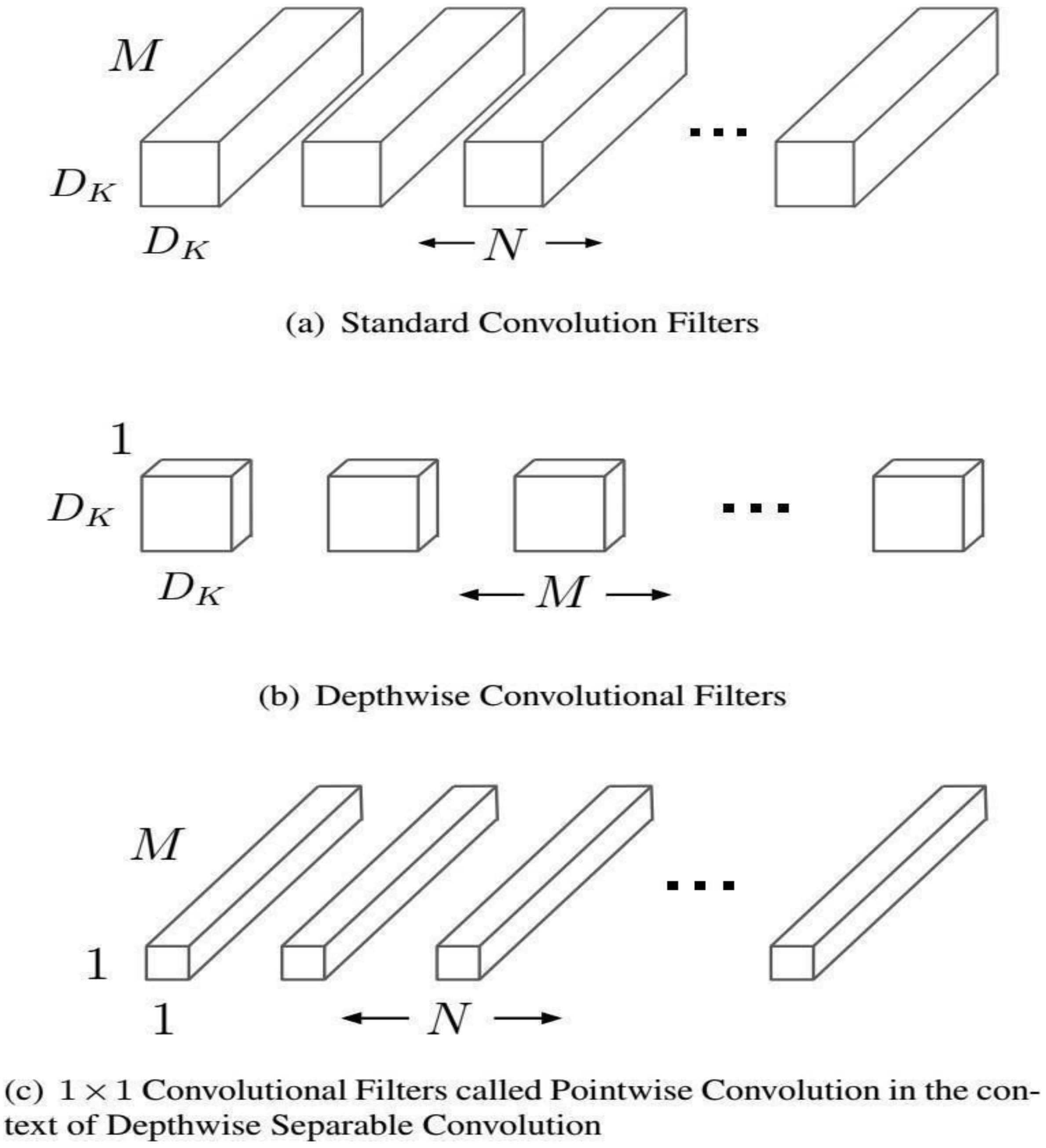

M
D_K
D_K
$\leftarrow N \rightarrow$
(a) Standard Convolution Filters
1
D_K
D_K
$\leftarrow M \rightarrow$
(b) Depthwise Convolutional Filters
M
1
1
$\leftarrow N \rightarrow$
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2: Filters

So the Mobilenet's total architecture is as follows: 30 layers, with 1 convolutional layer with stride 2 Layering based on depth. A pointwise layer that doubles the channel count Use stride 2 to create a depthwise layer A pointwise layer that doubles the number of channels, and so forth.

It also requires extremely little maintenance and thus performs admirably at highspeeds. Pre-trained models come in a variety of flavours, with the size of the network in memory and on disc proportional to the number of parameters utilised. The number of MACs

(Multiply-Accumulates), which is a measure of the number of fused Multiplication and Addition processes, is proportional to the network's speed and power consumption.

5.3 Training Overview

Designers will train hand motions in the coding part of this project. The finger related points are predicted using the vectors of hand-like properties. An application can be opened based on connected points.

CHAPTER 6
EXPERIMENTAL DESIGN AND EVALUATION

6.1 EXPERIMENTAL DESIGN

This project seeks to create a framework for implementing a system in Python to classify finger counts using a deep learning algorithm. Finger counts can be used to train the datasets.

6.2 EXPERIMENTAL RESULTS

The fraction of total number of flawless predictions to total amount of test data is used to calculate accuracy (ACC). 1 – ERR is another way to express it. The best possible accuracy is 1.0, while the worst possible accuracy is 0.0.

$$ACC = \frac{TP + TN}{TP + TN + FN + FP \times 100}$$

Equation 1

CHAPTER 7
MODEL OPTIMIZATION

7.1 OVERVIEW OF MODEL TUNING AND BEST PARAMETERS

Selection Image Generator

Create tensor image data in batches using real-time image augmentation. Then you can get a tf.data by executing image dataset from directory(main directory, labels='inferred').

A dataset containing batches of photos from the subdirectories class a and class b, as well as labels 0 and 1 (corresponding to class a and class b, respectively).
Jpeg, png, bmp, and gif are all supported image formats. The initial frame of animated gifs is truncated.

7.1.1 Mobilenet CNN

Mobile net is a model that uses convolution to filter images in the same way that CNN does, but in a different method than the preceding CNN. It employs the concepts of depth and point convolution, which differ from the usual convolution employed by conventional CNNs. As a result, CNN's ability to anticipate images improves, and they can now compete in mobile systems as well. Because these methods of convolution significantly cut comparison and recognition time, they provide a superior response in a short period of time, and we are adopting them as our image recognition model. The MobileNet models are simple to implement on mobile and embedded edge devices. MobileNets are built on a simplified design that builds low weight deep neural networks using depth-wise separable convolutions. To save calculation time and parameters, the MobileNet employs a Depthwise separable convolution rather than the standard convolution. It works by applying 1x1 convolution on each channel of the picture instead of as a block n times to generate n filters.

7.2 MODEL TUNNING AND EXPERIMENTS

Image Generator

- **Directory:** The data is stored in this directory. If labels is "inferred," it should have subdirectories with photos for each class. The directory structure is ignored otherwise.

- **Labels:** a list/tuple of integer labels the same size as the number of picture files found in the directory, None (no labels), or "inferred" (labels created from the directory structure). The alphabetic order of the picture file paths (obtained using os.walk(directory) in Python) should be used to sort the labels.
- **Label mode:** 'int': meaning the labels are encoded as integers (for example, sparse categorical crossentropy loss). - The labels are encoded as a category vector (e.g. for categorical crossentropy loss). - The labels (there can only be two) are encoded as float32 scalars with values of 0 or 1 (for example, binary crossentropy). - Null (no labels).
- **Lass names:** Only if "labels" is "inferred" is this option valid. This is a comprehensive list of class names (must match names of subdirectories). Controls the order in which the classes are taught (otherwise alphanumerical order is used).

7.2.1 TRAINING MODEL

Table 1: Training Model

input_shape	Optional shape tuple, only to be specified if include_top is Fals (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224 (with channels_first data format). It should have exactly 3 input channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value. Default to None. Input_shape will be ignored if the input_tensor is provided.
Alpha	Controls the width of the network. This is known as the width multiplier in the MobileNet paper. – If alpha < 1.0, proportionally decreases the number of filters in each layer. – If alpha > 1.0, proportionally increases the number of filters in each layer. – If alpha = 1, default number of filters from the paper are used at each layer. Default to 1.0.
depth_multiplier	Depth multiplier for depthwise convolution. This is called the resolution multiplier in the MobileNet paper. Default to 1.0.
Dropout	Dropout rate. Default to 0.001.

include_top	Boolean, whether to include the fully-connected layer at the top of the network. Default to True.
Weights	One of None (random initialization), ‘imagenet’ (pre-training on ImageNet), or the path to the weights file to be loaded. Default to imagenet.
Input_tensor	Optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model. Input_tensor is useful for sharing inputs between multiple different networks. Default to None.
Pooling	Optional pooling mode for feature extraction when include_top is False. None (default) means that the output of the model will be the 4D tensor output of the last convolutional block. Avg means that global average pooling will be applied to the outpu of the last convolutional block, and thus the output of the model wil be a 2D tensor. Max means that global max pooling will be applied.
Classes	The optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified. Defaults to 1000.
Classifier_activation	Str or callable. The activation function is to use on the “top” layer. Ignored unless include_top=True. Set classifier_activation=None to return the logits of the “top” layer. When loading pretrained weights, classifier_activation can only be None or “softmax”.
**kwargs	For backwards compatibility only.

CHAPTER 8

PRODUCT DELIVERY AND DEPLOYMENT

8.1 USER MANUALS

8.1.1 Need of the User manual:

Developers must definitely supply the product after developing the planned system. When the product is delivered, the user manual is required. Because they are end-users, the user has no past experience with the product. If a problem arises while using this product, the user can quickly resolve it by consulting this user manual.

8.1.2 Details that are available in the User manual:

The following are some of the details that can be found in the user manual: The user manual file contains the process of developing and deploying the product, as well as information on how to use it, such as demo screens and how to handle the main file, such as how to train the hand gesture datasets.

Because this product is built entirely with deep learning using tensor flow and KERAS CNN, not all machines can use deep learning libraries, which is why application errors can occur. Because such situations are common, the developer can provide a solution through the user handbook. The following are some of the problem-solving options in that manual: Error in the path Error reshaping CV2 blunder Error in the path: If the user is unable to select files from the correct path during the training phase, a path error may occur.

A reshaping error might occur if the input file size does not match the training imagefile size.

8.2 DELIVERY SCHEDULE

Hand gesture recognition from camera using deep learning can be deployed and deliverable with in the given timeline of product delivery

Table 2: Delivery Schedule

Date and No of Days	**Which one is Deliver the customer (user)**
First week (3-4 days)	Project Proposal
Second week	Designing
Third week	User accept design then develop code
Fifth week	Styling changes in Design
Sixth week	Run in local server
Seventh week	Complete design and Processing
Tenth week	Deployment and solving user Problem
Final Stage	Product Delivery in Market

CHAPTER 9

CONCLUSION

9.1 SUMMARY

An image processing technique is employed in this project to recognize the handmade movements. This application is used to present a modern integrated proposed system for voice-impaired people. This image processing approach allows blind people to engage with other blind people as if they were regular people. The user's data can be gathered with the use of a camera-based hand landmark. Each action will be significant in its own right. The gestures are more accessible than in the prior planned system. With the use of finger classification, the user is able to feel like a regular person. With enhanced accuracy, we may use a neural network model to classify finger counts and run apps based on count values.

9.1.1 LIMITATIONS:

There are certain flaws in this suggested system, which are listed below:

- Classify the finger counts in this proposed manner.
- This system may yield a false positive rate based on a variety of factors.
- As well as a rate of false positives in finger locations.

9.2 FUTURE WORK:

In the future, we can improve the technique by incorporating a deep learning algorithm into the framework and embedding the system with real-time embeddeddevices. Future enhancements will include additional research with the goal of creating abetter version

REFERENCES

[1] Kılıboz, Nurettin Çağrı, and Uğur Güdükbay. "A hand gesture recognition technique for human–computer interaction." Journal of Visual Communication and Image Representation 28 (2015): 97-104.

[2] Kılıboz, Nurettin Çağrı, and Uğur Güdükbay. "A hand gesture recognition technique for human–computer interaction." Journal of Visual Communication and Image Representation 28 (2015): 97-104.

[3] Shrivastava, Rajat. "A hidden Markov model based dynamic hand gesture recognition system using OpenCV." 2013 3rd IEEE International Advance Computing Conference (IACC). IEEE, 2013.

[4] Wang, Guojiang. "Facial expression recognition method based on Zernike moments and MCE based HMM." 2016 9th International Symposium on Computational Intelligence and Design (ISCID). Vol. 2. IEEE, 2016.

[5] Bagri, Neelima, and Punit Kumar Johari. "A comparative study on feature extraction using texture and shape for content based image retrieval." International Journal of Advanced Science and Technology 80.4 (2015): 41-52.

[6] Yun, Liu, Zhang Lifeng, and Zhang Shujun. "A hand gesture recognition method based on multi-feature fusion and template matching." Procedia Engineering 29 (2012): 1678-1684.

[7] Plouffe, Guillaume, and Ana-Maria Cretu. "Static and dynamic hand gesture recognition in depth data using dynamic time warping." IEEE transactions on instrumentation and measurement 65.2 (2015): 305-316.

[8] Chai, Xiujuan, et al. "Two streams recurrent neural networks for large-scale continuous gesture recognition." 2016 23rd International Conference on Pattern Recognition (ICPR). IEEE, 2016.

[9] Simão, Miguel A., Pedro Neto, and Olivier Gibaru. "Unsupervised gesture segmentation by motion detection of a real-time data stream." IEEE Transactions on Industrial Informatics 13.2 (2016): 473-481.

[10] Neverova, Natalia, et al. "Multi-scale deep learning for gesture detection and localization." European Conference on Computer Vision. Springer, Cham, 2014.

APPENDIX-A

A. DATASET

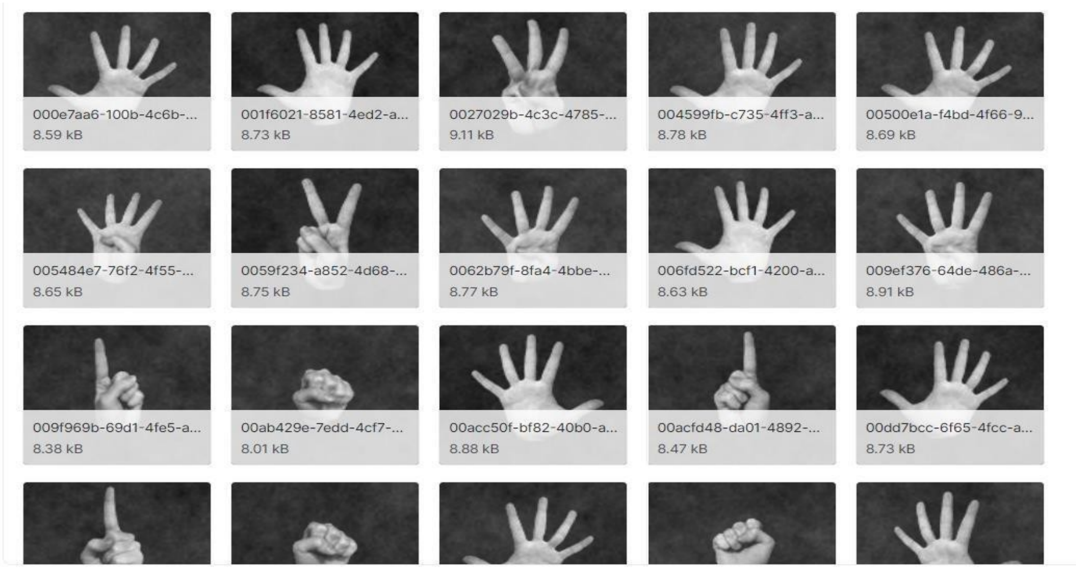
In this proposed system collect the variety of aircraft images with various size and types from various source.

Dataset

Found 16200 validated image filenames belonging to 6 classes.

Found 1800 validated image filenames belonging to 6 classes.

Found 3600 validated image filenames belonging to 6 classes.



APPENDIX-B

SOURCE CODE

In source code folder given below structure because in this proposed system deliver the product through windows application.

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dense, Activation

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.metrics import categorical_crossentropy

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import Model

from tensorflow.keras.applications import imagenet_utils

import os

import random

import matplotlib.pyplot as plt


trainpath = os.listdir("Data/fingers/train")

testpath = os.listdir("Data/fingers/test")


traindata = ['Data/fingers/train/' + i for i in trainpath]

testdata = ["Data/fingers/test/" + i for i in testpath]
```

```
traindata = pd.DataFrame(traindata, columns=['Filepath'])

testdata = pd.DataFrame(testdata, columns=['Filepath'])


traindata['target'] = traindata['Filepath'].apply(lambda a: a[-6:-5])

testdata['target'] = testdata['Filepath'].apply(lambda a: a[-6:-5])

#from IPython.display import Image

#Image(filename=traindata.Filepath[0], width=300,height=300)


ds_generator =
ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input,validati
on_split=0.1)


train_ds =
ds_generator.flow_from_dataframe(dataframe=traindata,x_col='Filepath',y_col='target',target_size=(22
4, 224),color_mode='rgb',class_mode='categorical',batch_size=16,subset='training')

val_ds =
ds_generator.flow_from_dataframe(dataframe=traindata,x_col='Filepath',y_col='target',target_size=(22
4, 224),color_mode='rgb',class_mode='categorical',batch_size=16,subset='validation')

test_ds =
ds_generator.flow_from_dataframe(dataframe=testdata,x_col='Filepath',y_col='target',target_size=(224,
224),color_mode='rgb',class_mode='categorical',batch_size=16)


mobile = tf.keras.applications.mobilenet.MobileNet()


mobile.summary()


x = mobile.layers[-6].output #Removing last 5 layers of mobilenet and addding softmax layer
```

```
output = Dense(units=6, activation='softmax')(x)

model = Model(inputs=mobile.input, outputs=output)


model = Model(inputs=mobile.input, outputs=output)


model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])


model.fit(train_ds, validation_data=val_ds, verbose=1, epochs=2)


model.summary()

print(model.summary())

model.evaluate(test_ds)


#model.save('CountFinger1.h5')

import cv2

import time

import pygame

import numpy as np

import mediapipe as mp

import matplotlib.pyplot as plt

from PIL import Image


# Initialize the mediapipe hands class.
```

```
mp_hands = mp.solutions.hands


# Set up the Hands functions for images and videos.

hands = mp_hands.Hands(static_image_mode=True, max_num_hands=2,
min_detection_confidence=0.5)

hands_videos = mp_hands.Hands(static_image_mode=False, max_num_hands=2,
min_detection_confidence=0.5)


# Initialize the mediapipe drawing class.

mp_drawing = mp.solutions.drawing_utils

from tensorflow.keras.models import Model




def detectHandsLandmarks(image, hands, draw=True, display=True):

    '''

    This function performs hands landmarks detection on an image.

    Args:

        image: The input image with prominent hand(s) whose landmarks needs to be detected.

        hands:  The Hands function required to perform the hands landmarks detection.

        draw:    A boolean value that is if set to true the function draws hands landmarks on the output
image.

        display: A boolean value that is if set to true the function displays the original input image, and the
output

                 image with hands landmarks drawn if it was specified and returns nothing.

    Returns:
```

```
    output_image: A copy of input image with the detected hands landmarks drawn if it was specified.
    results:      The output of the hands landmarks detection on the input image.
'''

# Create a copy of the input image to draw landmarks on.
output_image = image.copy()

# Convert the image from BGR into RGB format.
imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Perform the Hands Landmarks Detection.
results = hands.process(imgRGB)

# Check if landmarks are found and are specified to be drawn.
if results.multi_hand_landmarks and draw:

    # Iterate over the found hands.
    for hand_landmarks in results.multi_hand_landmarks:
        # Draw the hand landmarks on the copy of the input image.

        mp_drawing.draw_landmarks(image=output_image, landmark_list=hand_landmarks,
                                  connections=mp_hands.HAND_CONNECTIONS,
                                  landmark_drawing_spec=mp_drawing.DrawingSpec(color=(255, 255, 255),
                                                                               thickness=2, circle_radius=2),
                                  connection_drawing_spec=mp_drawing.DrawingSpec(color=(0, 255, 0),
                                                  thickness=2, circle_radius=2))



# Check if the original input image and the output image are specified to be displayed.
if display:

    # Display the original input image and the output image.
    plt.figure(figsize=[15, 15])
    plt.subplot(121);
    plt.imshow(image[:, :, ::-1]);
    plt.title("Original Image");
    plt.axis('off');
    plt.subplot(122);
    plt.imshow(output_image[:, :, ::-1]);
    plt.title("Output");
    plt.axis('off');

# Otherwise
else:

    # Return the output image and results of hands landmarks detection.
    return output_image, results
```

```
# Read a sample image and perform hands landmarks detection on it.

image = cv2.imread('media/sample.jpg')

detectHandsLandmarks(image, hands, display=True)

#Model.load_weights("./CountFinger1.h5")


def countFingers(image, results, draw=True, display=True):




    # Get the height and width of the input image.

    height, width, _ = image.shape


    # Create a copy of the input image to write the count of fingers on.

    output_image = image.copy()


    # Initialize a dictionary to store the count of fingers of both hands.

    count = {'RIGHT': 0, 'LEFT': 0}


    # Store the indexes of the tips landmarks of each finger of a hand in a list.

    fingers_tips_ids = [mp_hands.HandLandmark.INDEX_FINGER_TIP,
mp_hands.HandLandmark.MIDDLE_FINGER_TIP,

                        mp_hands.HandLandmark.RING_FINGER_TIP,
mp_hands.HandLandmark.PINKY_TIP]


    # Initialize a dictionary to store the status (i.e., True for open and False for close) of each finger of
```

```
both hands.
    fingers_statuses = {'RIGHT_THUMB': False, 'RIGHT_INDEX': False, 'RIGHT_MIDDLE': False,
'RIGHT_RING': False,
                        'RIGHT_PINKY': False, 'LEFT_THUMB': False, 'LEFT_INDEX': False,
'LEFT_MIDDLE': False,
                        'LEFT_RING': False, 'LEFT_PINKY': False}

    # Iterate over the found hands in the image.
    for hand_index, hand_info in enumerate(results.multi_handedness):

        # Retrieve the label of the found hand.
        hand_label = hand_info.classification[0].label

        # Retrieve the landmarks of the found hand.
        hand_landmarks = results.multi_hand_landmarks[hand_index]

        # Iterate over the indexes of the tips landmarks of each finger of the hand.
        for tip_index in fingers_tips_ids:

            # Retrieve the label (i.e., index, middle, etc.) of the finger on which we are iterating upon.
            finger_name = tip_index.name.split("_")[0]

            # Check if the finger is up by comparing the y-coordinates of the tip and pip landmarks.
            if (hand_landmarks.landmark[tip_index].y < hand_landmarks.landmark[tip_index - 2].y):
                # Update the status of the finger in the dictionary to true.
            fingers_statuses[hand_label.upper() + "_" + finger_name] = True

            # Increment the count of the fingers up of the hand by 1.
            count[hand_label.upper()] += 1

        # Retrieve the y-coordinates of the tip and mcp landmarks of the thumb of the hand.
        thumb_tip_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x
        thumb_mcp_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP - 2].x

        # Check if the thumb is up by comparing the hand label and the x-coordinates of the retrieved
landmarks.
        if (hand_label == 'Right' and (thumb_tip_x < thumb_mcp_x)) or (
            hand_label == 'Left' and (thumb_tip_x > thumb_mcp_x)):
          # Update the status of the thumb in the dictionary to true.
          fingers_statuses[hand_label.upper() + "_THUMB"] = True

          # Increment the count of the fingers up of the hand by 1.
          count[hand_label.upper()] += 1

    # Check if the total count of the fingers of both hands are specified to be written on the output image.
    if draw:
      # Write the total count of the fingers of both hands on the output image.
```

```
        cv2.putText(output_image, " Total Fingers: ", (10, 25), cv2.FONT_HERSHEY_COMPLEX, 1,
(20, 255, 155), 2)

        cv2.putText(output_image, str(sum(count.values())), (width // 2 - 150, 240),
cv2.FONT_HERSHEY_SIMPLEX,

                    8.9, (20, 255, 155), 10, 10)


    # Check if the output image is specified to be displayed.

    if display:


        # Display the output image.

        plt.figure(figsize=[10, 10])

        plt.imshow(output_image[:, :, ::-1]);

        plt.title("Output Image");

        plt.axis('off');


    # Otherwise

    else:


        # Return the output image, the status of each finger and the count of the fingers up of both hands.

        return output_image, fingers_statuses, count



camera_video = cv2.VideoCapture(0)
```

```
# Create named window for resizing purposes.
#cv2.namedWindow('Fingers Counter', cv2.WINDOW_NORMAL)

result = 0

c1 = 0
c2 = 0
c3 = 0
c4 = 0
c5 = 0
c6 = 0
c7 = 0
c8 = 0
c9 = 0
c10 = 0

# Iterate until the webcam is accessed successfully.
while camera_video.isOpened():

    # Read a frame.
    ok, frame = camera_video.read()
```

```
# Check if frame is not read properly then continue to the next iteration to read the next frame.
if not ok:
    continue

# Flip the frame horizontally for natural (selfie-view) visualization.
frame = cv2.flip(frame, 1)

# Perform Hands landmarks detection on the frame.
frame, results = detectHandsLandmarks(frame, hands_videos, display=False)

# Check if the hands landmarks in the frame are detected.
if results.multi_hand_landmarks:
    # Count the number of fingers up of each hand in the frame.
    frame, fingers_statuses, count = countFingers(frame, results, display=False)



    #print(sum(count.values()))
    result = sum(count.values())

    if(result==1):
        c1 += 1
        #print(c1)
        if(c1==20):
```

```
        #c1 = 0


        import subprocess as sp


        programName = "notepad.exe"
        sp.Popen([programName])
elif(result == 2):
    c2 += 1
    # print(c1)
    if (c2 == 20):
        # c1 = 0


        import webbrowser


        webbrowser.open("https://www.youtube.com/")
elif (result == 3):
    c3 += 1
    # print(c1)
    if (c3 == 20):
        # c1 = 0


        import webbrowser


        webbrowser.open("https://www.linkedin.com/")
else:
```

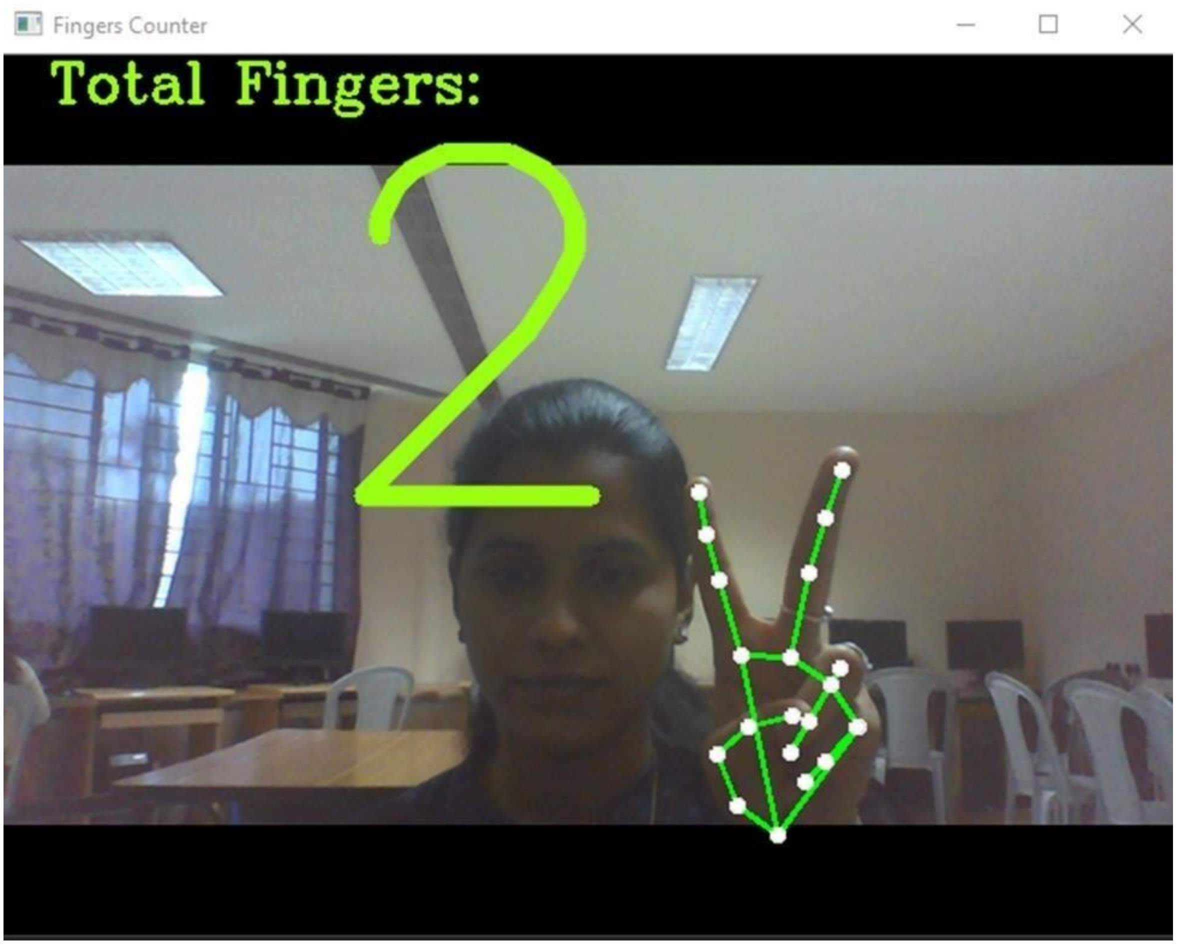
```
            cl  =0

            c2=0

            c3= 0









    # Display the frame.

    #imS = cv2.resize(frame, (960, 540))

    cv2.imshow('Fingers Counter', frame)


    # Wait for 1ms. If a key is pressed, retreive the ASCII code of the key.

    k = cv2.waitKey(1) & 0xFF


    # Check if 'ESC' is pressed and break the loop.

    if (k == 27):

        break


# Release the VideoCapture Object and close the windows.

camera_video.release()
```

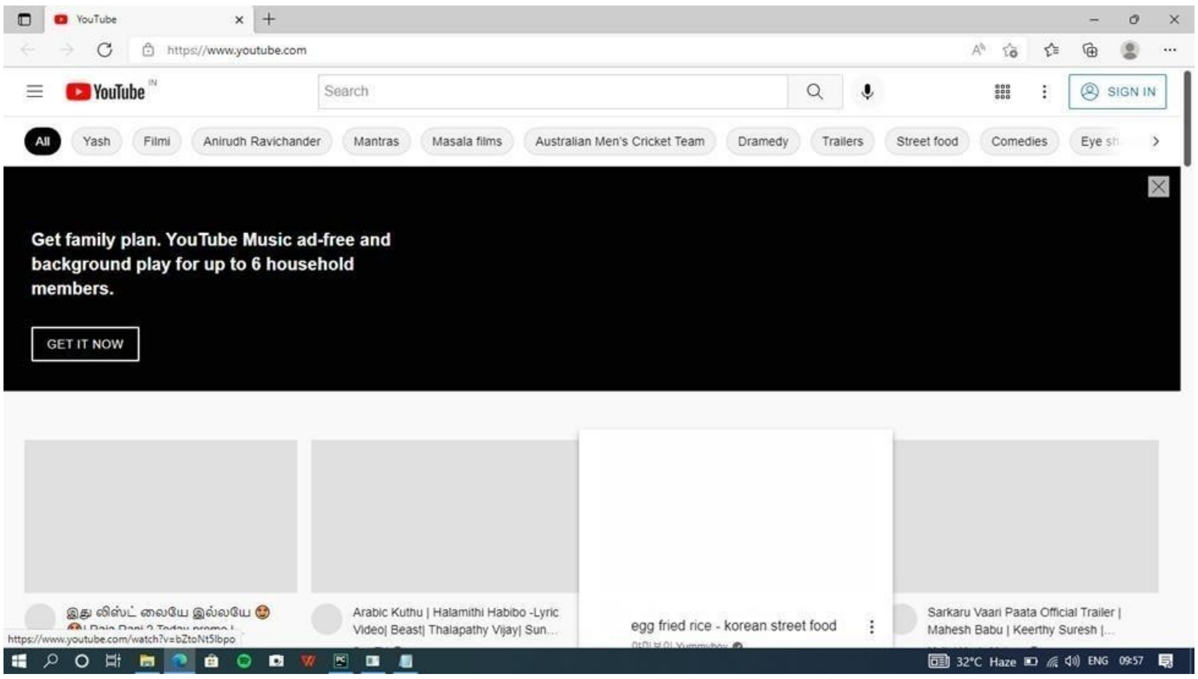
```
cv2.destroyAllWindows()
import mediapipe as mp
import cv2
import numpy as np
import uuid
import os


'''import subprocess as sp
programName = "notepad.exe"
#fileName = "sms.txt"
#sp.Popen([programName, fileName])
sp.Popen([programName])'''



mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
cap = cv2.VideoCapture(0)


with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        ret, frame = cap.read()


        # BGR 2 RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # Flip on horizontal
        image = cv2.flip(image, 1)

        # Set flag
        image.flags.writeable = False

        # Detections
        results = hands.process(image)

        # Set flag to true
        image.flags.writeable = True

        # RGB 2 BGR
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
       # print(results)

        # Rendering results
        if results.multi_hand_landmarks:
            for num, hand in enumerate(results.multi_hand_landmarks):
                mp_drawing.draw_landmarks(image, hand, mp_hands.HAND_CONNECTIONS,
                                        mp_drawing.DrawingSpec(color=(14, 22, 76), thickness=2, circle_radius=4),
                                        mp_drawing.DrawingSpec(color=(24, 44, 250), thickness=2,
circle_radius=2),
                                        )
```

```
    cv2.imshow('Hand Tracking', image)


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


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

APPENDIX-C

OUTPUT SCREENSHOTS





Get family plan. YouTube Music ad-free and background play for up to 6 household members.

இது லிஸ்ட் லையே இல்லயே 😍

Arabic Kuthu | Halamithi Habibo -Lyric Video| Beast| Thalapathy Vijay| Sun...

egg fried rice - korean street food

Sarkaru Vaari Paata Official Trailer | Mahesh Babu | Keerthy Suresh |...