Department of Informatics

Technical University of Munich

Master's Thesis

Hand pose estimation and gesture detection from webcam images

Ashish Khanal

Department of Informatics

Technical University of Munich

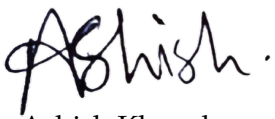
Master's Thesis

Hand pose estimation and gesture detection from webcam images

Hand- und Gestenerkennung in Webcam-Bildern

Author:	Ashish Khanal
Supervisor:	Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Deitrich
Submission Date:	November 15th, 2021

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.



November 15th, 2021 Ashish Khanal

Abstract

Hand pose estimation and gesture detection has been challenging but has garnered a lot of recognition recently as it allows an interesting way for us to interact with our devices. From its inception, gesture recognition techniques have been used to understand human gestures and body language, thereby building a bridge between humans and machines. This enables us to replace input devices such as keyboard and mouse, and interact naturally with the machine. This can significantly improve user experience and accessibility for people with special needs. This thesis is centered on hand gesture recognition, which is both a challenging and promising topic in the area of human-computer interaction. This study explores the detection of hand gestures and poses through webcam images, and compares two different implementations: classical approach and MediaPipe. The classical approach is based on the computer vision methods, where the image frames obtained from the webcam are subjected to filtering, detection and recognition. However, MediaPipe is a multimodal framework developed by Google to create Machine Learning pipelines for handling gesture recognition. From testing and experimental results it can be concluded that MediaPipe is more precise and faster than the classical approach, and can handle more complicated motions, poses and gestures, since it incorporates machine learning technologies under the hood.

Contents

Abstract iv

1 Introduction 1

2 State of the Art 4
2.1 Computer Vision . . . 4
2.1.1 Skin colour Filtering . . . 5
2.1.2 Morphological opening and closing . . . 8
2.1.3 Hand Gesture Detection . . . 8
2.1.4 Gesture recognition . . . 12
2.1.5 Background Subtraction . . . 13
2.2 MediaPipe Hands . . . 14
2.2.1 Datasets . . . 14
2.2.2 Palm Detection Model . . . 15
2.2.3 Hand landmark Model . . . 18

3 Hand pose estimation and gesture detection from webcam images 21
3.1 Data sets . . . 21
3.2 Evaluation Metric . . . 23
3.3 Hand gesture detection . . . 26
3.3.1 Classical method . . . 26
3.3.2 MediaPipe . . . 32
3.4 Experiment and Results . . . 35
3.4.1 Classical Approach . . . 35
3.4.2 MediaPipe . . . 38
3.4.3 Comparative Analysis of Classical and MediaPipe Approach . . . 41

4 Conclusions 44
4.1 Summary . . . 44
4.2 Discussion . . . 44
4.3 Outlook . . . 45

Glossary 50

Bibliography 50

1 Introduction

Hand pose estimation and gesture recognition has received great interests in the recent years, due to its natural human-computer interaction HCI characteristics. As technology has advanced, the disconnect between human and machine has narrowed significantly. In comparison to previous generations, the current generation is more reliant on these computer-based devices for day to day activities. As our dependence on electronic devices grow, there is a need to advance technology that connects humans and machines. One of such uses which has received great recognition and interests recently, is hand gesture recognition.

Gestures are expressive, meaningful body motions that involve physical movements of the fingers, hands, arms, head, face, or torso with the goal of communicating or engaging with the surroundings. Gestures are complementary to words in human-human interaction for those who can speak, while gestures are the only methods of communicating to other humans for those who do not have the ability to speak. Thus, hand gesture recognition makes it easier to regulate human-human or human-machine communication. Static gestures, in which the user takes a certain stance or configuration, or dynamic gestures, which include prestroke, stroke, and poststroke phases, are also possible. However, other gestures, such as those used in sign languages, have both static and dynamic features [18]. Thus, static and dynamic hand gesture recognition are both included in vision-based hand gesture recognition.

Human hand movement capture and estimation is an old computer vision problem that has been investigated since the 1990s. The introduction of glove-based control interfaces marked the beginning of hand gesture recognition for computer control. Researchers discovered that sign language-inspired movements may be utilized to provide simple orders for a computer interface. With the introduction of more precise accelerometers, infrared cameras, and fibreoptic based-sensors, progress in hand gesture recognition is slow. Some of these advancements in glove-based systems allowed for computer vision-based recognition without the need of any sensors on the glove [22].

Glove-based hand gesture recognition has challenges and, thus vision-based recognition needs to be developed. The glove-based technique employs specifically constructed gloves with a variety of sensor types and numbers to track finger flexion and contacts in real time. There are two major drawbacks of this method. To begin with, the sensors utilized in the gloves are quite expensive, thus the experiments are not accessible to the

general public. People would be irritated and uncomfortable as a result of the repeated use of gloves. Because the gloves will be used by a large number of individuals, additional precautions should be taken to prevent the spread of skin disease. That is why, vision based approaches were developed to tackle such drawbacks.

Vision-based approaches are not flawless in and of itself, since huge position variations, a highly articulated structure, major self-occlusions, perspective shifts, and data noises make it tough to solve [20]. Furthermore, in most applications, real-time performance is required. However, because today's devices have sufficient processing capacity and computer vision libraries are readily available online, anybody can conduct hand gesture detection at their comfort.

Hand gesture recognition has been used in many fields of study including recognition of sign language, which is beneficial for people with disabilities [27], and in medicine to comprehend a user's movements in real time and modify items in a medical data visualization environment [29]. One study [11] explored the application of this approach in 3D gaming, specifically the Candy Crush game. Another study [6] used a real-time hand gesture to control a robot to facilitate human-robot interaction. In another study, gesture control is used to control TV operations and interact with a mobile device without the need of sensors attached to the human body [25]. Finally, one research [13] looks at how gesture interaction technology may be used in virtual reality (VR), along with discussion of the present challenges with gesture interaction.

Recently, Facebook's launch of Metaverse has revolutionized the area of augmented reality (AR) and gesture detection. The metaverse, according to Meta's message, is a place where you may virtually meet, work, and play utilizing a VR headset, glasses, or your mobile. In the AR environment, the user's avatar is utilized to interact with the avatars of other users. This paves the way for more gesture control research and applications in AR. The user is able to convey their sentiments and emotions virtually through their avatar. As a result, not only hand gesture recognition but also face gesture recognition would be investigated further, and this would be the future of gesture recognition and control in general.

This thesis explores the classical computer vision models for hand pose estimation and compares it with a deep neural network solution called MediaPipe. Summary of the classical approach is given in Figure (1.1). Google's open-source MediaPipe was initially released in July 2019. MediaPipe, an open-source framework for complicated perception pipelines based on accelerated inference (e.g., GPU or CPU), currently provides quick and accurate, but independent, solutions for these jobs [8]. It uses Single shot detection (SSD), an object detection algorithm to detect the palm and extrapolates the information from this to detect the fingers using hand landmark model. There are classical computer vision techniques available now that detect hand gestures and positions, however, these meth-

ods have not been as accurate as desired. Therefore, the goal of this study is to explore the computer vision and neural network solutions for hand gesture detection.

This thesis is divided into three main chapters. The first chapter is the literature reviews of the theoretical background and state of the art hand gesture detection techniques, mainly self implemented computer vision-based technique and MediaPipe. The second chapter discusses the method used in detection process and the results obtained. Finally, the last chapter concludes the thesis by summarizing the key findings of the study, discussing the main results and provides an outlook on the improvements that can be made in the future.

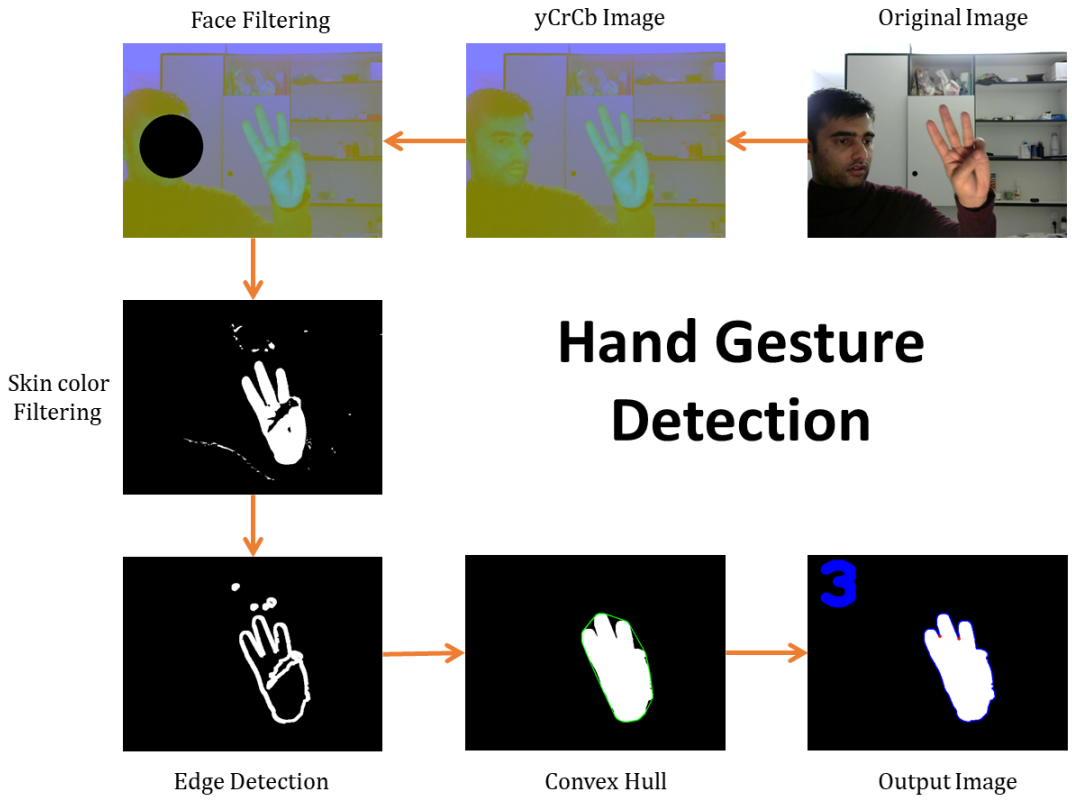


Figure 1.1: Summary overall pipeline for Hand Gesture Detection.

2 State of the Art

Hand gesture detection and posture estimation have garnered a significant interests in recent times. There are various classical computer vision methods that have been used to detect hand gestures, but it has not been accurate and precise enough to be put into widespread use. However, with the introduction of fast and accurate object detection algorithms like Single Shot Detection (SSD), the accuracy and speed of hand gesture estimation has great promise to improve. In this section, we will review the state of the art for the classical computer vision algorithms and the modern algorithms such as MediaPipe that uses deep learning for hand gesture detection. The gesture recognition can be classified in several categories as described in Table (2.1).

Table 2.1: Hand gesture recognition system classification, Table from [12].

Category	Type
Application	Sign Language, Robot Control, Tracking Gesture, Games
Motion	Static, Dynamic
Image acquired data	Camera(s), Video, Data Glove Instrumented Device
Data dimensions	2D, 3D
Number of hands used	One hand, two hand
Input features	3D Hand Model, Appearance Based, Low Level Features
Gestures modality	Communicative, Manipulative

2.1 Computer Vision

Gesture recognition is the study of the use of human gestures to convey information for command and control purposes. HCI involves the tracking and interpretation of movement as commands. There are variety of computer vision methods that can be used for gesture detection without the use of expensive and complicated glove-based sensors. Some tasks necessitate more expensive cameras and depth sensors, while others can be accomplished with just a webcam. A study by Oudah et al. [21] presents the review of different computer vision techniques and the required hardware currently used for hand gesture detection such as hand gesture detection based on instrumented glove approach, skin colour segmentation using glove markers and so on. Hand gesture recognition through computer vision, based on skin colour segmentation, consist of the following general steps:

2.1.1 Skin colour Filtering

The first step in the hand gesture detection is the skin colour filtering to extract the skin pixels from the image. One of the main aspects of concern is the colour space when extracting this feature. There are several colour spaces that can possibly be used for this task and some colour spaces works better than others. Human skin color has a limited spectrum of hues and is not thoroughly saturated. According to research, the colour of skin is created by a combination of blood (red) and melanin (brown, yellow). Thus, human skin color is grouped in a small area within a specific color space. Several studies have found that the most significant variation in skin color across people is due to intensity rather than chrominance. Skin-like regions can easily be spotted if a picture is first converted into a color space that separates the luminance channel and two chrominance components [1].

Below, are the different colour spaces and their skin detection equations. RGB colour space consists of the Red, Green and Blue channels. Normalized RGB colour space is formed using the RGB values of an image. Equations (2.1) to (2.3) are used compute the Normalized RGB values from the RGB values [23].

$$R' = \frac{R}{R+G+B} \tag{2.1}$$

$$G' = \frac{G}{R+G+B} \tag{2.2}$$

$$B' = \frac{B}{R+G+B} \tag{2.3}$$

And the corresponding equations to detect the skin pixels are given by Equations (2.4) to (2.6) [7].

$$\frac{R'}{G'} > 1.185 \tag{2.4}$$

$$\frac{R'G'}{(R'+G'+B')^2} > 0.107 \tag{2.5}$$

$$\frac{R'G'}{(R'+G'+B')^2} > 0.112 \tag{2.6}$$

Another color space that is commonly used is the HSV color space. It has channels Hue(H), Saturation(S) and Value(V). H and S represents the chrominance while V represents the luminance. HSV can be computed from RGB using the Equations (2.7) to (2.9).

$$V = max(R', G', B') \tag{2.7}$$

$$S = \begin{cases} \dfrac{V - min(R', G', B')}{V}, & \text{if } V \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

$$H = \begin{cases} \dfrac{60(G' - B')}{V - min(R', G', B')}, & \text{if } V = R \\ 2 + \dfrac{60(B' - R')}{V - min(R', G', B')}, & \text{if } V = G \\ 4 + \dfrac{60(R' - G')}{V - min(R', G', B')}, & \text{if } V = B \\ \text{if } H < 0, \text{then } H = H + 360 \end{cases} \tag{2.9}$$

Then, the skin pixels are given by the Equations (2.10) to (2.12) [23].

$$V \geq 40 \tag{2.10}$$

$$0.2 < S < 0.6 \tag{2.11}$$

$$0 < H < 25 \quad \text{or} \quad 335 < H < 360 \tag{2.12}$$

Another colour space that can be used is the yCrCb colour space. This space is similar to HSV colour space, and separates the luminance and chrominance and is used for skin detection. Y is luminance and the Cr and Cb represent the red difference and blue difference respectively. yCrCb colour space can be obtained from RGB values using Equations

(2.13) to (2.15).

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{2.13}$$

$$Cr = 0.713(R - Y) + delta \tag{2.14}$$

$$Cb = 0.564(B - Y) + delta \tag{2.15}$$

where,

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit image} \\ 0.5 & \text{for floating-point images} \end{cases}$$

The threshold that is used for yCrCb colour space to compute the skin pixels are given by the intersection of Equations (2.16) to Equation (2.18) [23].

$$Y > 80 \tag{2.16}$$

$$135 < Cr < 180 \tag{2.17}$$

$$85 < Cb < 135 \tag{2.18}$$

The output of the edge detection will be a binary image with the skin pixels as white and the rest of the pixels as black [Figure 2.1].

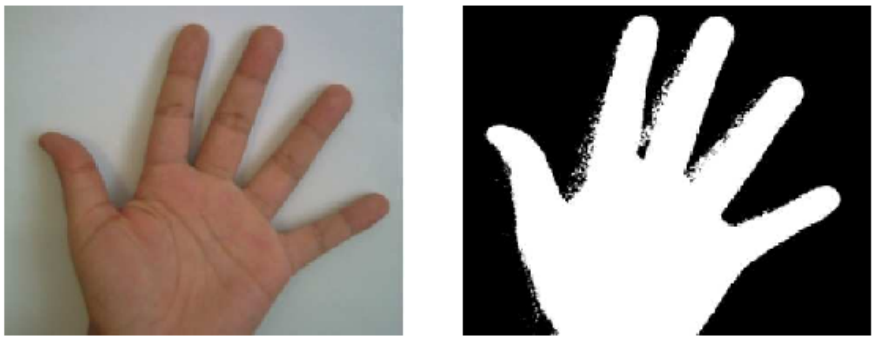


Figure 2.1: Output of skin colour filtering, Figure from [24].

2.1.2 Morphological opening and closing

Morphological opening and closing are the operators used in computer vision on the binary image for manipulation of the input image to obtain a clearer image. Opening and Closing both rely on the erosion and dilation operator. Erosion increases the background area and shrinks the foreground of the image while dilation increases the foreground and shrinks the background area. Opening is the erosion of the input image followed by the dilation while closing is the dilation of input image followed by the erosion [19].

The structuring element (SE) is the mask that has a specific shape and size and is used to probe the input image. Same structuring element is used for the erosion and dilation during opening and closing. Different structural elements can be chosen based on the problem at hand [19].

Once the erosion is carried out for the opening process the image cannot be recovered but the dilation is done to recover as much as possible. Therefore, opening is used to remove small objects from the foreground and add them to the background. During morphological closing, small holes in the foreground are filled as dilation is done first which increases the foreground regions before eroding them [19].

2.1.3 Hand Gesture Detection

After skin filtering, there are a variety of ways to detect just the hand. There are few ways in which hand can be detected and we will discuss these steps below. Some studies [24] and [4] use edge detection and template matching while another study [3] detect the palm and then segment the fingers to detect the hands.

1. **Edge Detection**
   The process of edge detection is usually focused on locating pixels where the image brightness has a discontinuity. This enables the detection of objects within the particular region and allows the computation of various properties of that object such as perimeter or area [24]. An edge pixel is defined by two characteristics, its edge strength which is equal to the magnitude of the gradient and the edge direction which is equal to angle of the gradient. For a discrete function, a gradient is not defined; rather, the gradient, which can be defined for an ideal continuous image, is inferred using various operators such as Roberts, Sobel, and Perwitt [3].

   Robert Filter is a 2×2 gradient mask that considers the diagonal element of the pixels. The gradient magnitude estimation of a function is given in Equation (2.19) where G is the gradient and f is the function.

$$\begin{aligned} G[f[i,j]] &= |f[i,j] - f[i+1,j+1]| + |f[i+1,j] - f[i,j+1]| \\ G[f[i,j] &= |G_x| + |G_y| \end{aligned} \tag{2.19}$$

G_x and G_y are the convolution masks given in Table 2.2 and 2.3 [26].

Table 2.2: Robert (G_x)

1	0
0	-1

Table 2.3: Robert (G_y)

0	-1
1	0

Sobel operator puts emphasis on the pixel that is in the center of the mask. It is one of the most commonly used edge detectors. It can be represented by a 3×3 convolutional mask. The magnitude of the gradient is given by (2.20).

$$M(x, y) = \sqrt{S_x^2 + S_y^2}. \tag{2.20}$$

Gradient operators S_x and S_y are the convolution masks given in Table 2.4 and 2.5 [26].

Table 2.4: Sobel (S_x)

-1	0	1
-2	0	2
1	0	1

Table 2.5: Sobel (S_y)

1	2	1
0	0	0
-1	-2	-1

Perwitt Operator is used for detecting horizontal and vertical edges. It is similar to the Sobel filter but it doesn't put emphasis on the center pixel of the mask. The convolutional mask representing the Perwitt operator is given in Table 2.6 and 2.7 [26].

Table 2.6: Perwitt (P_x)

-1	0	1
-1	0	1
1	0	1

Table 2.7: Perwitt (P_y)

1	1	1
0	0	0
-1	-1	-1

2. **Palm and finger detection**
   Instead of edge detection, another procedure to detect the hand is by first detecting the palm and then finding the fingers. The steps to first detect the palm and then the finger are based on another study [3].

   Given the binary image output of the skin colour filtering, the first step would be to find the palm point of the hand. The palm point can be found by using the distance

transform method.

Distance Transform is a image processing technique done on a binary image. There are two types of pixels; background pixels and foreground pixels. The area of interest in our case is the foreground pixel, which is made up of hand pixels. There are two types of neighborhood connections: four connected and eight connected. When calculating the distance transform, we must account for the distance metric. The choices that we make can have a significant impact on the output [30]. For the hand gesture recognition, city block distance was chosen. As shown in Figure 2.2, it is a simple distance transform metric (four neighbourhood algorithm). As we propagate from the edge of the image, the center pixel will have the highest value, and the pixels with the highest value will be our palm point [3].

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0
0	1	2	2	2	2	2	1	0
0	1	2	3	3	3	2	1	0
0	1	2	3	4	3	2	1	0
0	1	2	3	3	3	2	1	0
0	1	2	2	2	2	2	1	0
0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0

(b)

Figure 2.2: City Block Distance transform on (a) binary image (b) distance transform, Figure from [3].

Taking the palm point as a center, a circle is drawn. As long as hand pixels are detected, the radius of the circle is gradually increased. This will form an inner circle of maximal radius, which contains the palm of the hand. The palm point, as shown in Figure 2.3, is the center of that circle. After that, the palm mask needs to be computed. The algorithm to compute the palm mask and wrist points is obtained from a previous study [3], and the result after applying the algorithm can be seen in Figure 2.3.

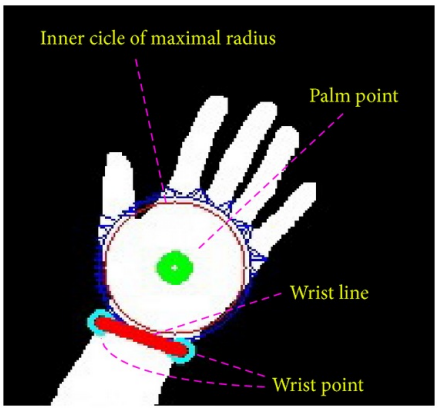


Figure 2.3: Results after finding the palm point, inner circle with maximal radius, wrist points, and the wrist lines, Figure from [3].

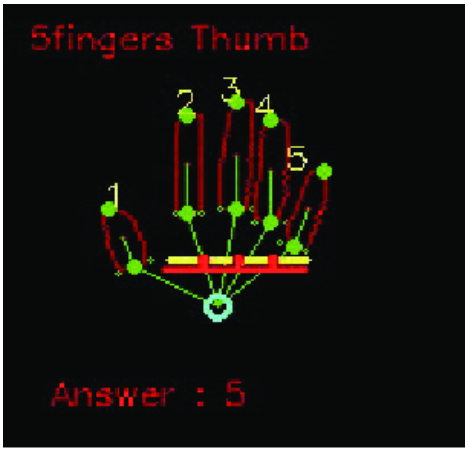


Figure 2.4: Yellow line is the palm line and red line parallels the wrist line, Figure from [3].

After finding the wrist points and a palm point, an arrow can be drawn from the palm point to the center of the wrist line. The arrow will be modified so that it will always be pointed to the north. The hand image is then rotated concurrently to make it invariant to rotation of the hand [3].

The finger and palm can be segmented easily with the palm mask as any hand pixels that does not belong to the palm mask are segmented as fingers and the rest as the palm. After the segmentation, regions of certain size are picked, and a minimal bounding box is drawn around it. The center of the bounding box is represented as

the center of the finger [3].

Now, the lines can be drawn from the center point of the finger to the palm point. The angle between these lines and the wrist line is then calculated, and if it is less than $50°$, it is concluded that the thumb is included in the image. However detection of other fingers needs extra steps. First, palm line which is parallel to the wrist line is found. This can be done by starting from the row of pixels consisting of the wrist line and moving upwards. If, for each row of these pixels, the intersection between the line and the hand pixels contains all connected white regions, then, it moves upwards until the disconnected regions are found. As the disconnected regions indicate the separation of fingers, we will have found our palm line at the bottom of the fingers. If, the thumb exists in the pixels, there will be a break in this process as the thumb is disconnected from the other regions. To solve this, only the edge points of the palm are considered to find the intersection with the line. As the center of the fingers are found in the earlier step, it is then used to detect each finger. The palm line is divided into four parts for each finger. If the horizontal co-ordinates of the center of the finger falls into the first part then it is the fore finger, if it falls into second part then it's the middle finger, third part is the ring finger and fourth part is the little finger. The image of this process is given in Figure 2.4 [3].

2.1.4 Gesture recognition

Finally the hand gestures can be detected by using a classifier. For both cases, simple rule classifiers can be used to detect the number of fingers. The steps required to complete the detection include:

1. **Gesture recognition after edge detection** After the edge detection process, the edges of the hand are obtained. Convex hull is then used to find the convexity defect. This is a simple process that can be used to count the number of fingers that have been raised but it does not make the distinction between which fingers have been raised.

   Convex Hull of a set S can be defined as the smallest convex set that contains the edge [24]. Convex set contains line segments between any two points in the set. This is shown in the Figure 2.5. Left image is convex as any line segments drawn between the two points will be contained with in the set whereas for the non-convex set, there will be some points in the set where the line segments are not contained inside of it. Therefore, the convex hull algorithm eliminates all the concave regions. The result will be a polygon that encapsulates the whole hands [24].

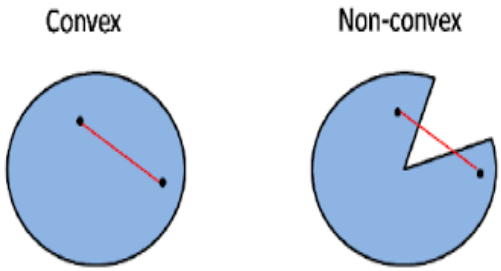


Figure 2.5: Convex set in the left and non-convex set in the right, Figure from [2].

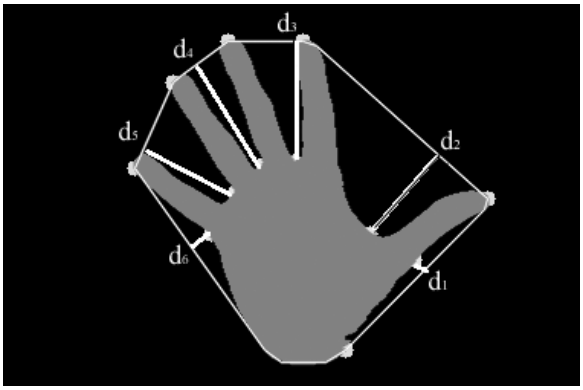


Figure 2.6: Convexity defects and the depths d_i, Figure from [32].

The convexity defect is then calculated to figure out the number of elevated fingers. Convexity defect is the farthest point from the convex point. In this case, finger tips would be the convex point and the trough between the fingers would be the convexity defect. The number of elevated fingers can be computed by using the number of convexity defects, defect depths and depth average. The convex hull, convexity defect and depth (d_i) are shown in Figure 2.6 [24].

2. **Gesture recognition after finger and palm segmentation** In this process, the Hand Gestures are simply classified after using a rule classifier. As the specific fingers, are detected in an earlier step, the detected fingers can be deducted the specific gestures can be identified [3].

2.1.5 Background Subtraction

This is an optional pre-processing step that can be added after skin colour segmentation, depending on the problem at hand. Background subtraction method such as frame differencing is used to differentiate the foreground pixels from the background pixels. Frame

differencing is a background subtraction method that involves subtracting one picture from another and labeling any change that is large enough as foreground. This will ensure that all the moving body parts will be captured, however, if the lighting conditions change rapidly, the pixel value will shift as the light intensity changes, and additive noise will contribute to the output. This can be done simply by subtracting one frame from the other and is given by the Equation (2.21) [4].

$$\text{Diff}_n = \text{img}_{n+1} - \text{img}_n. \tag{2.21}$$

2.2 MediaPipe Hands

MediaPipe hands provides the user with the ability to do real time hand tracking with just a normal webcam. It uses machine learning to infer 21 3D landmark from a single image. It is a two step process that consists of the palm detection model and the hand landmark model that can track multiple hands in real time [17]. The two models and the datasets used to train are described in details below.

2.2.1 Datasets

To train the MediaPipe models, three different data sets were chosen. The datasets used to train the model are described below.

- In-the-wild dataset: This dataset comprises six thousand images with a wide range of characteristics, such as geographical diversity, lighting conditions, and hand appearance. The dataset's limitation is that it does not include complicated hand articulation.
- Googles in house collected dataset: This collection contains ten thousand images that depict all physically conceivable hand movements from various perspectives. The dataset's shortcoming is that it was compiled from only 30 people with little variance in their backgrounds. The in-the-wild and in-house datasets are excellent complements for improving robustness.
- Synthetic dataset: This dataset maps a high-quality synthetic hand model to the corresponding 3D coordinates after rendering it over diverse backdrops. A commercial 3D hand model with 24 bones and 36 blendshapes was used to regulate the thickness of the fingers and palm. The hand model also includes five textures with various skin tones. Transformation video sequences between hand positions were constructed and hundred thousand images were sampled from the videos. Each posture was rendered using three separate cameras and a random high-dynamic-range lighting environment.

In-the-wild dataset was used for the palm detector since it is sufficient for hand localization and has the most variability in appearance. All datasets, however, are utilized to train the hand landmark model [35].

2.2.2 Palm Detection Model

Detection of hands is a complex task compared to others as it has to work across variety of hand sizes and be able to detect occluded part of hands. The contrast and shapes of noses and lips could be used as distinct features in a face detection task. However, such features are missing in the hand detection task, making it much more difficult. As a result, the palm detector is trained because estimating bounding boxes around rigid objects like the palm and fist is a much easier task. Then, encoder decoder feature is used for a larger scene context awareness similar to FPN and finally focal loss is minimized to support large scale variance [35].

The object detection module currently consists of the following steps : Hypothesize bounding boxes, resample features for each box and apply a high-quality classifier. There are several object detection algorithms that are currently being used such as Faster RCNN, SSD, You only look once (YOLO) detector. But they all differ on speed and accuracy. The speed of object detection is measured in frames per second (FPS) and accuracy is often determined by calculating Mean Average Precision (mAP). There's a tradeoff between speed and accuracy and usage scenarios vary depending on the problem at hand. Faster R-CNN has high accuracy but only runs at 7 frames per second. It could be applicable in a scenario where accuracy is of utmost importance. For example, it is not fast enough for automated Driving as speed and accuracy both matter and faster RCNN is very slow. Likewise YOLO is known for its speed, however, it suffers in terms of accuracy. Unlike other deep neural network based object detectors, SSD forgoes the resampling of pixels and features for bounding box hypothesis. SSD is much faster and more accurate (59 FPS with mAP 74.3%) than YOLO(45 FPS with mAP 63.4%) and Faster RCNN (7 FPS with 73.2% mAP) for the VOC2007 test. (Liu 2). SSD produces high accuracy even on low resolution input images which improves the speed vs accuracy trade off [16].

1. **Single shot detector (SSD)**
   The process for SSD includes creating the bounding box around each object in the image. This is known as the Ground Truth (GT). Then, the whole image is divided into a feature map of different sizes. Default boxes of different sizes are evaluated at each square of the feature map. These default boxes consists of two information; the location ($\mathrm{center_x}$, $\mathrm{center_y}$, width, height) and the confidence for all object categories ($c_1, c_2, ..., c_p$). The shape offsets and confidences for all object categories should be predicted for each default box. The default boxes should be matched to the GT boxes during training.

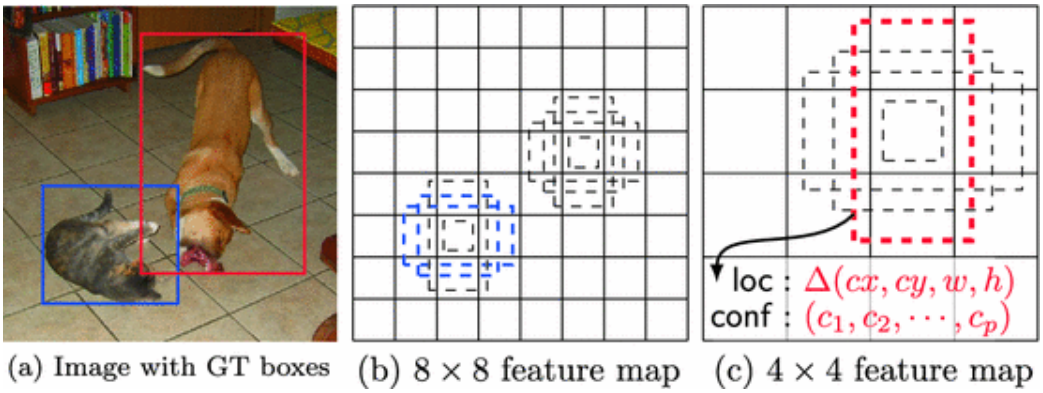
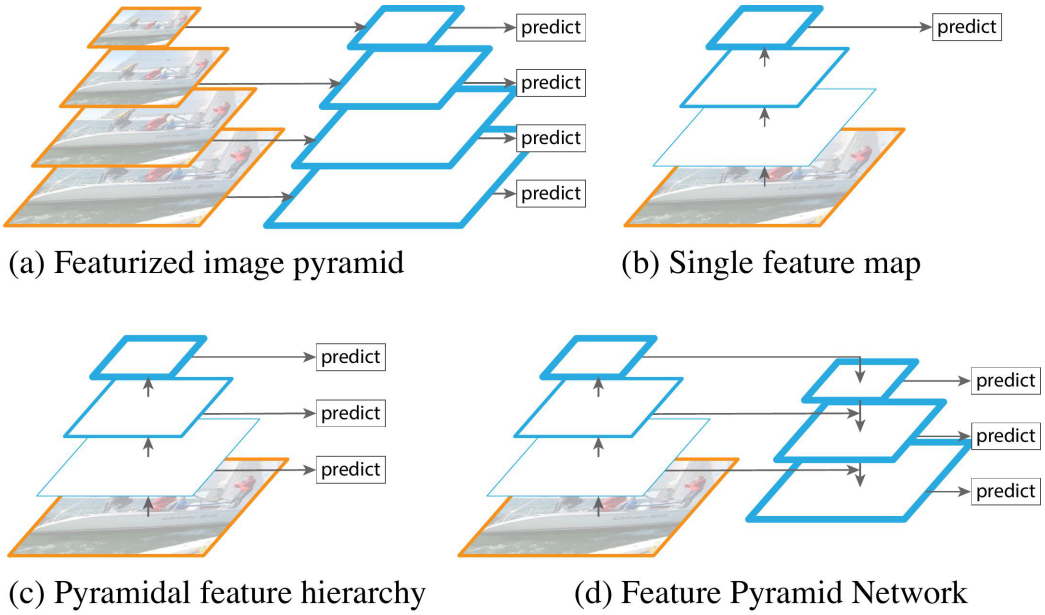


Figure 2.7: Single Shot Detector (SSD) framework, loc is location of the bounding box, conf is the confidence of all object categories, Figure from [16].

The boxes that match the default box should be considered positive, while the rest should be considered negative. The model loss should be calculated as the weighted sum between localization loss (Smooth L1) and confidence loss (e.g softmax) [16]. In Figure 2.7, the example of an image with cats and dogs are given. The feature map chosen is of size 8×8 and 4×4. The default boxes that match up with the Ground truth box of cats and dogs are treated as positives and the rest as negatives. SSD Model is based on the feed forward convolutional network that produces a fixed size collection of bounding boxes and gives confidence value for the object that exists in the bounding box [16].

2. **Feature Pyramid Network (FPN)**
   FPN is used as neck to the backbone of SSD. FPN is used to compute the multi-feature representation of the image. The featured pyramid are built upon the image pyramid and are scale inavariant. These are used to capture the various sizes of objects. Figure 2.8 (a) image is downsampled and feature maps are computed at each level. This is denoted by the blue outline which gets thicker for stronger features. Figure 2.8 (b) is a typical convolution neural network type structure that downsamples the image and then predicts at the end. This is expected to be fast as it only uses single scale features as compared to (a). Figure 2.8 (c) is used by SSD. This architecture uses pyramidal feature hierarchy computed by convolutional neural network as if it were featurized image pyramid. Architecture in Figure 2.8 (d) is proposed in [14] as it is as fast as (b) and (c) but more accurate. This architecture first downsamples the image to it's lowest resolution and then upsamples them. The features are then combined by the lateral connection which is $1 * 1$ convolution. This combines the features from highest resolution to lowest resolution so it makes it more accurate than (b) and (c) due to availability of more information [14].



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 2.8: Feature Pyramid Network (FPN), Figure from [14].

3. **Minimising focal loss**
   Focal loss addresses the problem of one stage detector where there is an imbalance between the foreground mage and the background image. With cross entropy loss, even easy examples that are easily classified incur a loss that is greater than expected. And if these small losses are summed over a large number of easy examples, then, the added loss can overwhelm the harder examples. The cross entropy loss is overwhelmed by the huge class imbalance experienced during dense detectors. The majority of the loss is made up of easily recognized negatives, which dominate the gradient. Balanced cross entropy addresses the class imbalance by introducing a weighting factor alpha. This helps to distinguish between the negatives and positives but does not help with easy and hard examples. Easy examples are those examples that have a probability of ground truth class($p_t >> 0.5$). The equation for balanced cross entropy is given in (2.22) where CE is the cross entropy loss [15].

$$CE(p_t) = -\alpha \log(p_t) \tag{2.22}$$

   Focal loss reshapes the loss function to down weight the easy examples and focuses the training on hard negatives.The equation for focal loss(FL) is given in (2.23) where y is the smoothing parameter [15].

$$FL(p_t) = -(1 - p_t)^y \log(p_t) \tag{2.23}$$

In Equation (2.23), misclassified hard example has smallp_t, the modulating factor $(1 - p_t)$ is close to 1. Therefore the loss function is unaffected. But for easy examples, p_t goes to 1, modulating factor goes to 0, the loss for well classified easy examples is down weighted. The focusing parameter y is used to smoothly adjust the rate at which easy examples are down weighted [15]. In practice, alpha variant focal loss (Equation (2.24)) is used for better accuracy over Equation (2.23) [15].

$$FL(p_t) = -\alpha_t(1 - p_t)^y \log(p_t) \tag{2.24}$$

2.2.3 Hand landmark Model

Following palm detection throughout the whole picture, our hand landmark model uses regression to conduct exact land-mark localization of 21 2.5D coordinates inside the identified hand areas. This is represented in Figure 2.9 Even with partially visible hands and self-occlusions, the model develops a consistent internal hand posture representation. The output given by the hand landmark models are the 21 3D landmarks, hand flag to know the probability of hand presence and the handedness to know which hand is present left or right. This architecture is given in 2.10. The topology similar to Multiview bootstrapping used in [28] is used to detect the 21 landmarks specifically to boost the performance of a keypoint detector. This helps the hand landmark model by estimating the keypoints of the occluded hands. And to recover from the tracking failure a model similar to [10] is generated. This is done in order to increase the likelihood that a suitably aligned hand is present in the given crop [35].

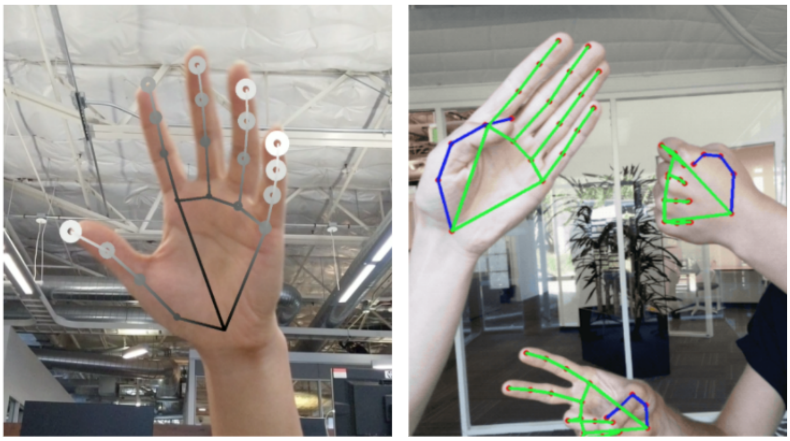


Figure 2.9: Example of hand pose estimation using MediaPipe, figure from [35].

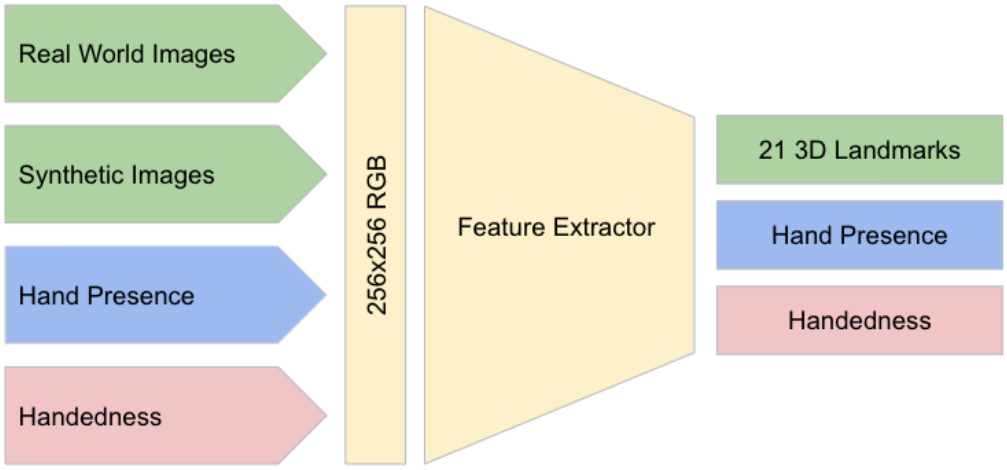


Figure 2.10: Architecture of hand landmark model for MediaPipe, figure from [35].

1. **Multiview Bootstrapping**
   Even if there is severe occlusion in a single picture of the hand, there is usually an unobstructed view. Multiview bootstrapping systematizes this concept, resulting in a more effective hand detector that generalizes beyond the capture scenario. A weak detector trained on a small, annotated dataset can localize subsets of key-points in good views and filter out incorrect detection by using 3D triangulation [28]. The algorithm for this is given in [28].

2. **Recover tracking failure**
   The algorithm to recover tracking failure for face detection is given in [10]. MediaPipe uses a similar method to make sure that the hand is present in the cropped image. We will breifly look at the image processing pipeline given in [10] for the face mesh prediction and generalize it to hand scenario.

   A lightweight hand detector can be used to process the image from the camera input, producing hand bounding rectangles and a number of landmarks for the wrists, and fingers. The landmarks can be used to rotate a hand rectangle such that, the line linking the wrist center is aligned with the rectangle's horizontal axis [10].

   Original image is cropped according to the rectangle obtained in the previous step and resized to provide the input to the mesh prediction neural network (varying in size from 256×256 pixels in the full model to 128×128 pixels in the smallest one). 3D landmark coordinate vector is generated by the model and it is transferred back into the original image coordinate system. The probability of a reasonably aligned hand

being present in the supplied crop is produced via a distinct scalar network output [10].

3 Hand pose estimation and gesture detection from webcam images

Hand pose estimation and gesture recognition is a common problem with a variety of solutions. Hand pose estimation is the process of determining where the fingers and joints of the hands are located, whereas gesture recognition is the process of translating a human hand gesture into a mathematical form. Glove-based approaches marked the beginning of hand gesture recognition, but they were out of reach for the general public due to the high cost of hardware and the risk of disease spreading due to glove sharing. As a result, vision-based techniques needed to be developed so that they could be widely used by the general public. Various vision-based techniques have been developed over the years that are more accessible but still require expensive hardware. In this thesis, only webcam images are used to investigate a traditional approach using computer vision algorithms and a convolutional neural network approach provided by Google called MediaPipe, so that gesture detection can be done with minimal hardware such as a smart phone's front camera or laptop's webcam.

This chapter is divided into four sections. The first section describes the data sets and the experimental setup. The second section describes the evaluation metric that is used to evaluate the gesture recognition systems. The, third section describes both gesture recognition approach using classical computer vision techniques and neural network approach using MediaPipe. Finally, in the last section, results are described using the evaluation metric for both approaches.

3.1 Data sets

Classical computer vision method proposed in this project, can detect up to six simple gestures, but MediaPipe can recognize far more complex gestures. For this project, there are six gestures that will be detected depending on the number of elevated fingers in the hand. As only the elevated fingers in the hands are being considered, it does not matter which fingers are being raised. The gesture recognition system does not need to distinguish between the fingers that are being raised. As a result, there are six classes, ranging from zero to five raised fingers.

Six gestures that are used for the evaluation of hand gesture detection are given in Figure 3.1.

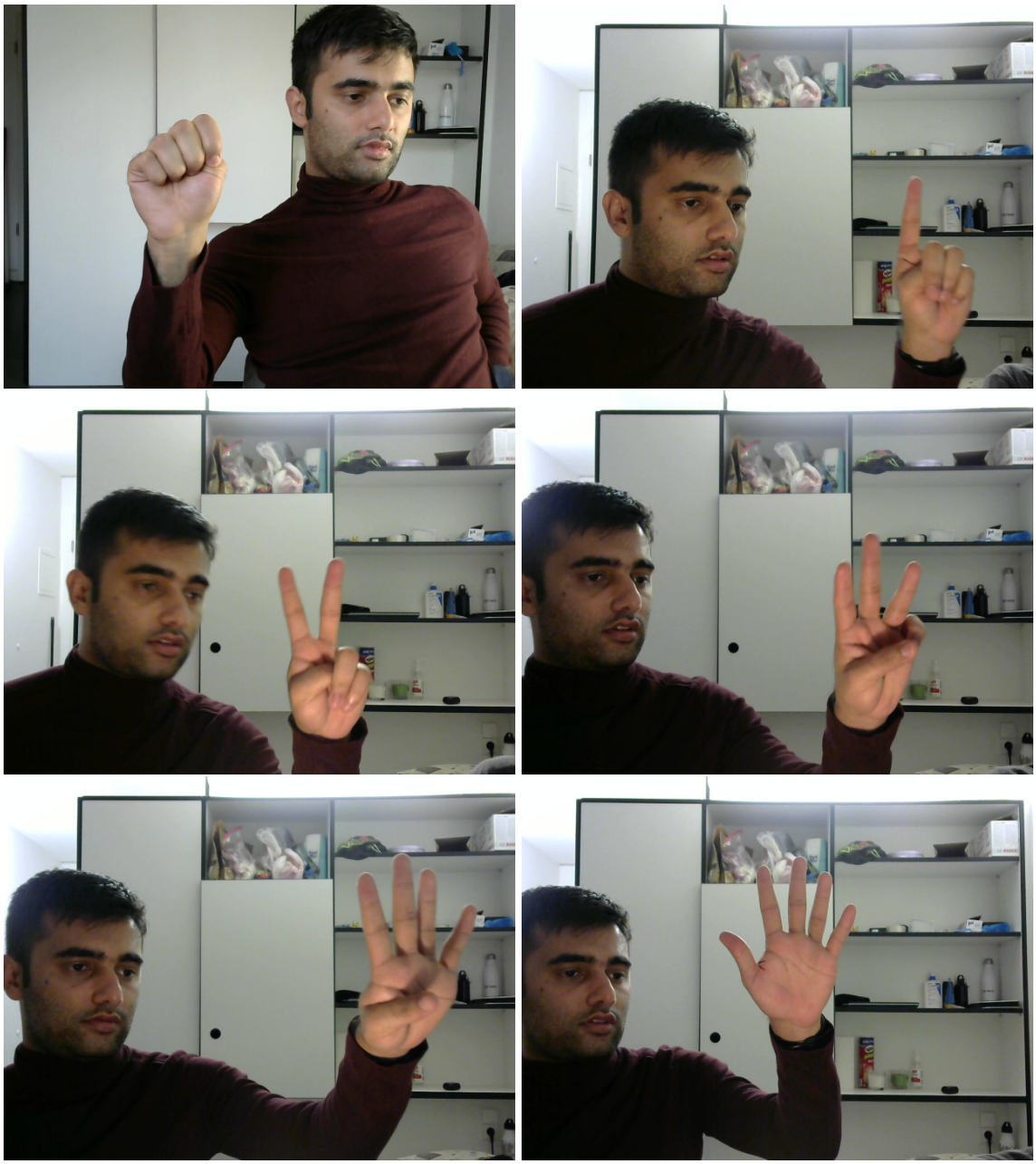


Figure 3.1: Example of Input Image data for 6 classes.

In the data collection phase, the images were initially taken from the webcam of the laptop. Since the images taken were of poor quality due to noise, the data was collected again from the Logitech C270 webcam. The images were taken in a variety of lighting conditions. For each class, five hundred images were taken, half of which were taken in daylight and

half of which were taken at night with the room's normal lightbulb. The hands' angles and positions were also changed. Both approaches for hand gesture detection will use the same data set, and the results will be calculated.

3.2 Evaluation Metric

The number of elevated fingers in one hand is counted for gesture recognition. That means there are six possible outcomes: zero fingers raised, one finger raised and so on. As a result, this is a multi-class classification problem with six distinct classes. This task must be changed from multiclass classification to binary classification because the majority of the metrics are defined for the binary classification problem. This can be done in one of two ways: one vs one (OVO) or one vs rest (OVR) classification.

OVO separates the multi-class problems into several binary classification task for each pair of classes. For example, our classification task will be separated into 0 vs 1, 0 vs 2, 0 vs 3 and so on. The problem with this strategy is the computational workload. Each pair of classes require a separate binary classifier and total number of binary classifier is given in Equation (3.1), where N is the number of classes and S is the total number of classifiers.

$$S = \frac{N(N-1)}{2} \tag{3.1}$$

OVR creates an individual classifier for each target class. A single class is chosen and marked as positive while the rest are marked as negative. The classifiers that were built for this study are given in Table (3.1).

Table 3.1: Classifiers for one vs rest.

0	or not	0
1	or not	1
2	or not	2
3	or not	3
4	or not	4
5	or not	5

This strategy lowers the computational cost as compared to OVO classifier but can significantly make the classification task imbalanced. To calculate the metric of the classifier a simple arithmetic mean of the metric of all classes was used. The metric that was used to evaluate the classifier accuracy, precision, recall and f1 score are discussed later. Firstly, True positives, True Negatives, False positive and False Negative were defined. The definitions are provided in Table 3.2 and example of definition for class label 3 is given in Table 3.3.

Table 3.2: Confusion matrix

	Actual positives	Actual negatives
Predicted positives	True positive (tp)	False Positive (fp)
Predicted negatives	False negative (fn)	True Negative (tn)

Table 3.3: Example for class 3, which can be interpolated to other classes.

tp	Ground truth is 3 and Predicted is 3
fp	Ground truth is negative but predicted is 3
tn	Ground truth is negative image and predicted is negative
fn	Ground truth is 3 but Predicted is any other class(0,1,2,4,5)

The above definition can then be used to define the metrics. The definition of all the metrics are given in Table 3.4

Table 3.4: Metrics used in the evaluation.

Metrics	Formula	Description
Accuracy	$\frac{tp+tn}{tp+tn+fp+fn}$	Most commonly used method for well balanced classification task. This measures the number of correct predictions made by the classifier.
Precision	$\frac{tp}{tp+fp}$	Measures the positive labels that are correctly predicted out of all actual predicted labels.
Recall	$\frac{tp}{tp+fn}$	Measures the positive labels that were correctly predicted out of all predicted positive result
F1-score	$\frac{2\times(precision\times recall)}{precision+recall}$	Takes into consideration the precision and recall. It is a good metric as there exists tradeoff between precision and recall

The metrics were computed using the OVR method. For this, all images from each class are used in the calculations. Each class had a total of five hundred images. Therefore, three thousand images must be evaluated for each classifiers listed in Table 3.1. For a particular class, true positives made up five hundred of that class's images, while true negatives made the other two thousand and five hundred images from the remaining classes. As a result, the true positives and true negatives images used were unbalanced, but this imbalance were consistent across all classifiers. If the classification metric was hampered by this imbalance, it would have affected all classes equally.

As there were six classifiers, an averaging technique is necessary to get a single metric for all classifiers so that two gesture recognition approaches can be compared. To do this, an average of the classifiers was found. Averaging techniques are described below.

- *Macro averaged metric:* This is used for the balanced classification tasks. This is a simple arithmetic mean of a particular metric of all classes. It gives equal weights to all the classes. If the classes are imbalanced weights can be calculated by dividing the number of occurrences of that class by total number of images. The weights can then be multiplied with all the metrics for e.g. precision and accuracy [31].

- *Micro averaged Metric:* Micro averaging is summing up all the true positives and the true negatives across all classifiers and dividing it by the total number of images in all classifier. This metric is similar to accuracy [31].

Another analysis tool that can be used is the Receiver Operating Characteristics (ROC) curve. ROC curve is a widely used curve for analysis of binary classification. For this, True positive rate (TPR) and False positive rate (FPR) must be calculated at different detection threshold. Then, ROC curve is plotted using FPR in the x-axis and TPR in the y axis .

Equations (3.2) and (3.3) are used to calculate TPR and FPR. TPR is the same as recall and gives number of true positive result for all positive samples in the evaluation data set, whereas FPR gives the number of false positive results for all actual negative samples in the evaluation data set.

$$TPR = \frac{tp}{tp + fn} \tag{3.2}$$

$$FPR = \frac{fp}{fp + tn} \tag{3.3}$$

TPR and FPR are computed for several detection thresholds. Due to this, only the ROC curve for MediaPipe can be computed. There is no detection threshold available for the classical computer vision method. Even though, this metric can't be used to compare the two gesture detection task it is still a very useful metric to see how well each classifier for MediaPipe performed. The TPR and FPR values can then be plotted against each other. Example of ROC curve is given in Figure (3.2). The red line indicates the functioning of a random classifier. Anything above the red line indicates the classifier being better than the random classifier while anything below indicates the classifier being worse than the random classifier. The perfect classifier is at $TPR = 1$ and $FPR = 0$ represented by blue dot in Figure(3.2). This can be used to figure out the best detection threshold. The threshold value that maximizes the TPR and minimizes the FPR can then be chosen as the detection threshold.

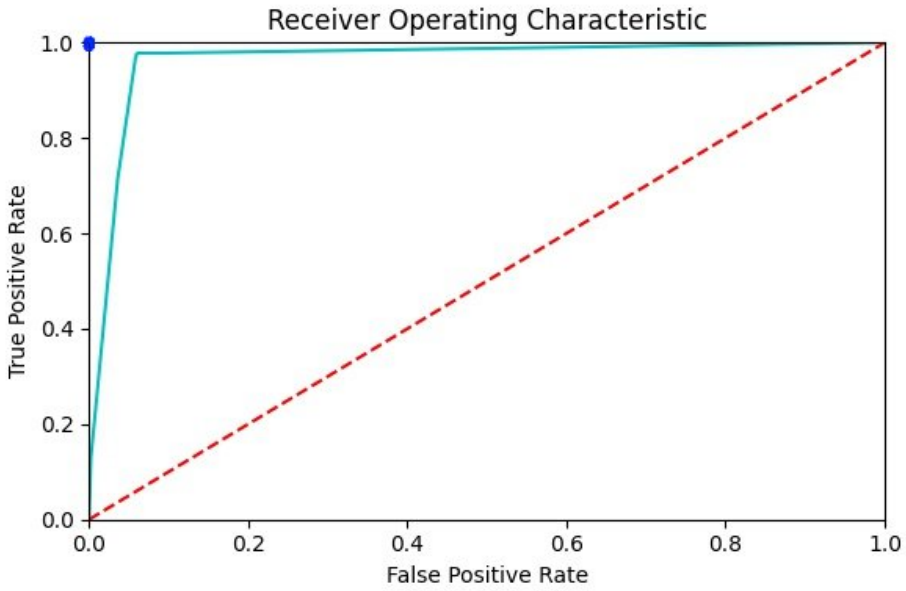


Figure 3.2: Example of an ROC curve where the red line indicates how a random classifier would work, blue dot represent the perfect classifier while the cyan is the ROC curve.

3.3 Hand gesture detection

In this section, methods for hand gesture detection using traditional computer vision algorithms and MediaPipe are discussed. The input of the gesture recognition is the RGB image while number of elevated fingers (0, 1, 2, 3, 4, 5) is the output. The classical approach is described first, followed by the MediaPipe approach.

3.3.1 Classical method

This section outlines the processes involved in applying the classical technique.

Step 1: Data collection

The input was captured with camera. The dimension of each image is 640×480. The collected data sets were for 0, 1, 2, 3, 4 and 5 fingers.

Step 2: Skin color filtering

The first process in the classical detection was the skin colour filtering method. There are variety of skin colour filtering method as described in Section (2.1.1). The yCrCb color

space was chosen because of its simplicity and the fact that it separates the chroma and luminance components. The RGB image was first converted to yCrCb colour space using Equation (2.13), (2.14), (2.15). Then, skin pixels were extracted from the image using Equation (2.16), (2.17),(2.18) . The result of the skin colour filtering can be seen in Figure 3.3. The skin filtered image not only consists of the hand but also the face and the noise in the background.

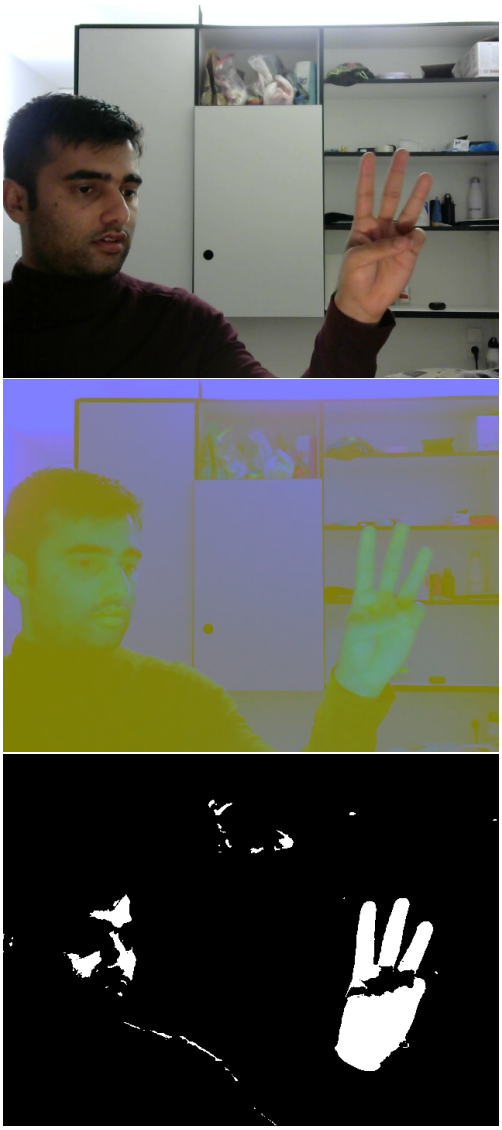


Figure 3.3: Original image on the top, image in yCrCb colour space on the middle and image after applying skin colour filtering on the bottom.

Step 3: Face removal

The next step in this approach was to detect the face so that it can be excluded. Haar cascade classifier was used to detect the face and blacked out so that the prominent contour remaining is of the hand. A Haar classifier, also known as a Haar cascade classifier, is a machine learning object detection program that can recognize objects in images and videos. It was used in the world's first real-time face recognition system. The classifier has four stages of operation. To begin, it computes Haar features, which are essentially calculations performed on adjacent rectangular regions at a specific location in a detection window. The calculation entails adding up the pixel intensities in each region and then subtracting the sums. Edge, line, and four rectangle features are among the Haar features. Then, it creates integral images which are sped up calculation of the Haar features. Then, in the next step Adaboost training is performed where Adaboost basically selects the most useful features and trains the classifiers to use them. Finally, by using cascading classifiers, the weak learners are merged into a strong learner to create a strong classifier. This causes the algorithm to either detect that an object has been located or to go on to the next region [33]. Open CV's already trained Haar cascade classifier was used to detect the face. The pre-trained Haar cascade classifier can be found in Open CV's github repository [5].

Result of applying the face pixels and blacking out the face is given in Figure 3.4. There are Morphological closing and opening done to get rid of some of the background noises as can be seen in Figure 3.4.

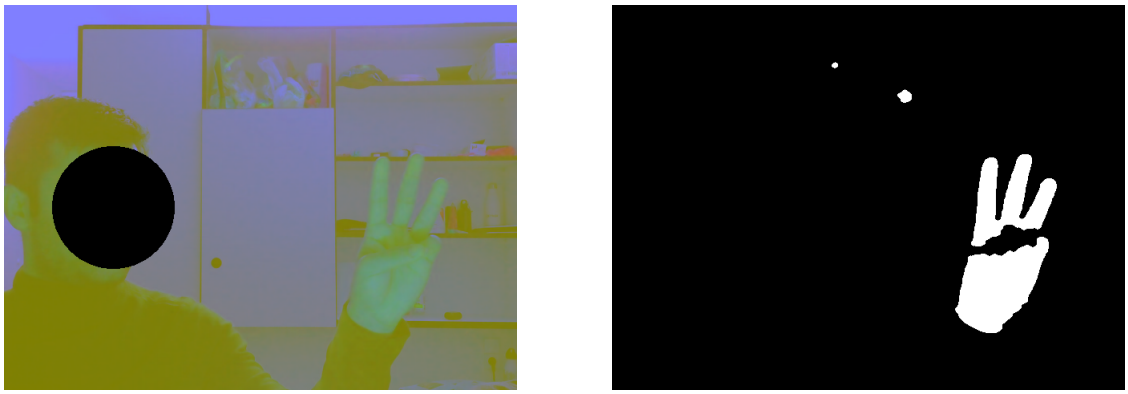


Figure 3.4: The output after applying face detection and cutting out the face pixel on the yCrCb image (Left). The output after applying skin filtering on the left image (Right).

Step 4: Edge detection

After the face was cut out, the prominent contour left on the image was of the hand, edge detection algorithm was applied to detect the edges of the hand. Edge detection algorithms are described in Section (2.1.3). The sobel operator was used for the edge detection and result after applying the sobel operator is given in Figure 3.5.

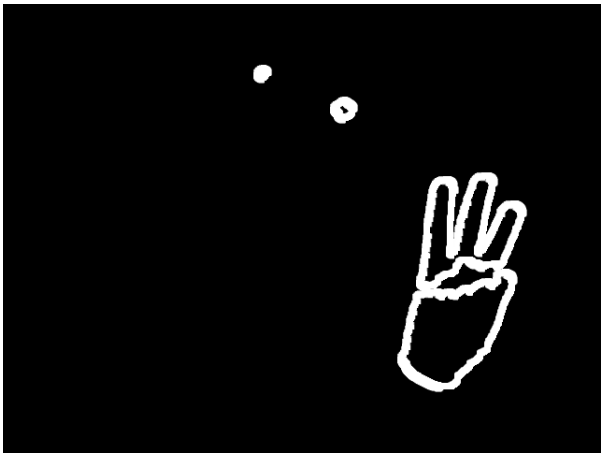


Figure 3.5: Applying sobel operator to binary image after skin color detection.

As we can see in Figure 3.5, only edges are detected so the standard flood fill algorithm was used to fill the hand with white pixels. Flood fill algorithm is a region filling algorithm. It is based on either a four or eight neighbourhood system. Flood fill algorithm takes a seed inside a closed area, such as our hand's edges, and recursively discovers all the pixels inside the known pixel's related region. The pixel is filled once it has been discovered [9].

Step 5: Contour Extraction

There were more skin pixels or noise in the image background. As shown in Figure (3.5), this resulted in numerous contours in the image. Maximum contour was taken to eliminate the unwanted contours in the background. As the hand was the most prominent object in the image, all noises and skin pixels identified in the background were eliminated. This is one of the limitation of this technology since the hand that has to be recognized must be in the foreground.

The contours of the hand were then detected from the binary inverted flood filled image. `cv2.findContours()` is a feature of Open CV's image processing that detects changes

in image color and marks them as contours. After the processed image was passed in the `cv2.findContours()` function, multiple contours were detected by the algorithm. Out of these, the maximum contour was filtered out according to their area. This gave out contours in the hand edges, because in the previous image processing steps, Haar classifiers were used to remove the face from the filtered image.

Step 6: Convex Hull and Convexity Defects

Convex hull of the image was computed. Convex hull encapsulates the hand with the polygon. As a result, Convex Hull formed a tightly fitted convex boundary around the hand. Any deviation of the hand from the convex hull is its convexity defect. The convex hull can be drawn by passing contours to the function `cv2.drawContours`. In order to determine the convexity defect of the max contour, the `cv2.convexityDefects()` function in OpenCV is utilized. This function accepts the contour and its related hull indices as input and outputs an array of convexity defects. This function returns an array with [**Start point**, **End point**, **Far point**, **Depth**] in each row where, **Start Point** is the first appearance of the convexity defect on the contour, **End Point** is the convexity defect ends at this point on the contour, **Far Point** is the farthest point within the defect from the convex hull and **Depth** is the approximate distance between the outermost points of the convex hull and the contour's farthest points. For a single convexity defect, we can derive the three sides a, b and c, where a is the distance measure between start and end point, b is the distance measure between start and far point and c is the distance measure between end and far point. Then, by using cosine theorem, the angle between the two fingers γ was calculated as

$$\gamma = cos^{-1}(\frac{a^2 + b^2 - c^2}{2\ a\ b}). \tag{3.4}$$

This can be visualized in Figures 3.7 and 3.8.

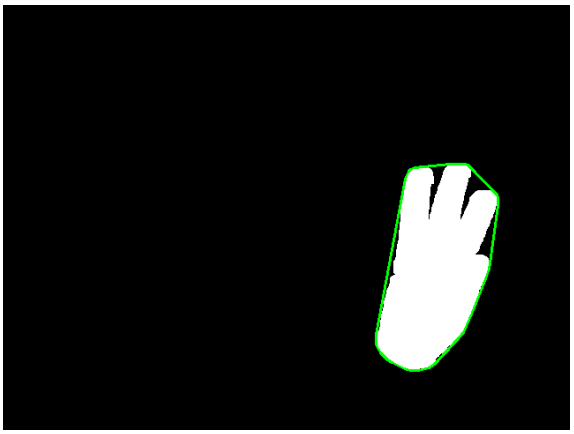


Figure 3.6: Convex hull of an image.

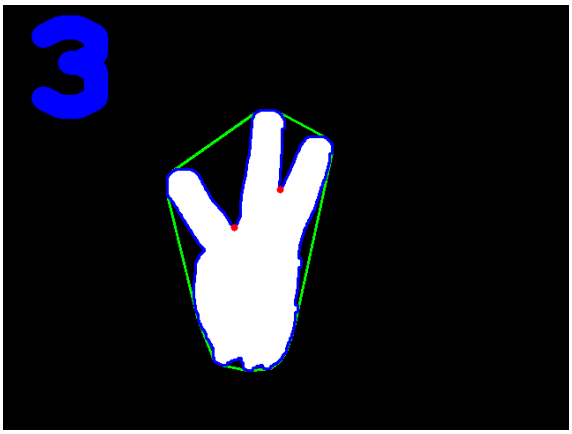


Figure 3.7: Convexity defect.

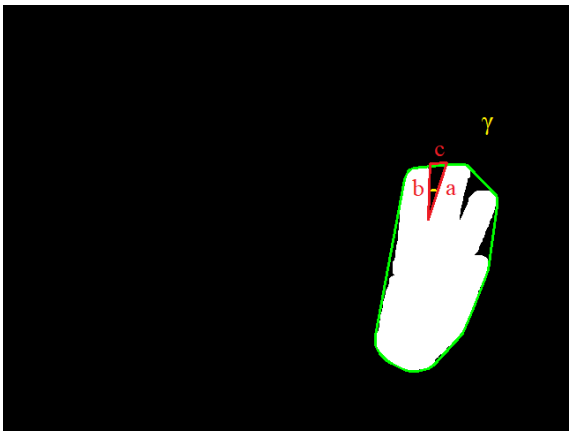


Figure 3.8: convexity defect with angle γ.

Step 7: Gesture Detection

After calculating the angle (as shown in Figure 3.8), the final step was to count the number of fingers. If the angle γ was less than 90^0, then, it was considered a finger. After determining γ, a circle was drawn with radius 4 in the approximate distance to the farthest point. A count was maintained, and finally, text was inserted into the image to reflect finger count. Figure 3.7 contains the convex hull on the left and the convexity defect and the gesture recognition on the right. If the convexity defect is not detected, finger tip detection is used to determine whether the number of elevated fingers is 0 or 1, whereas Equation (3.3.1) indicates the number of elevated fingers ranging from two to five.

$$\text{Number of fingers} = \text{number of defects} + 1$$

This entire process is shown in Figure 3.9.

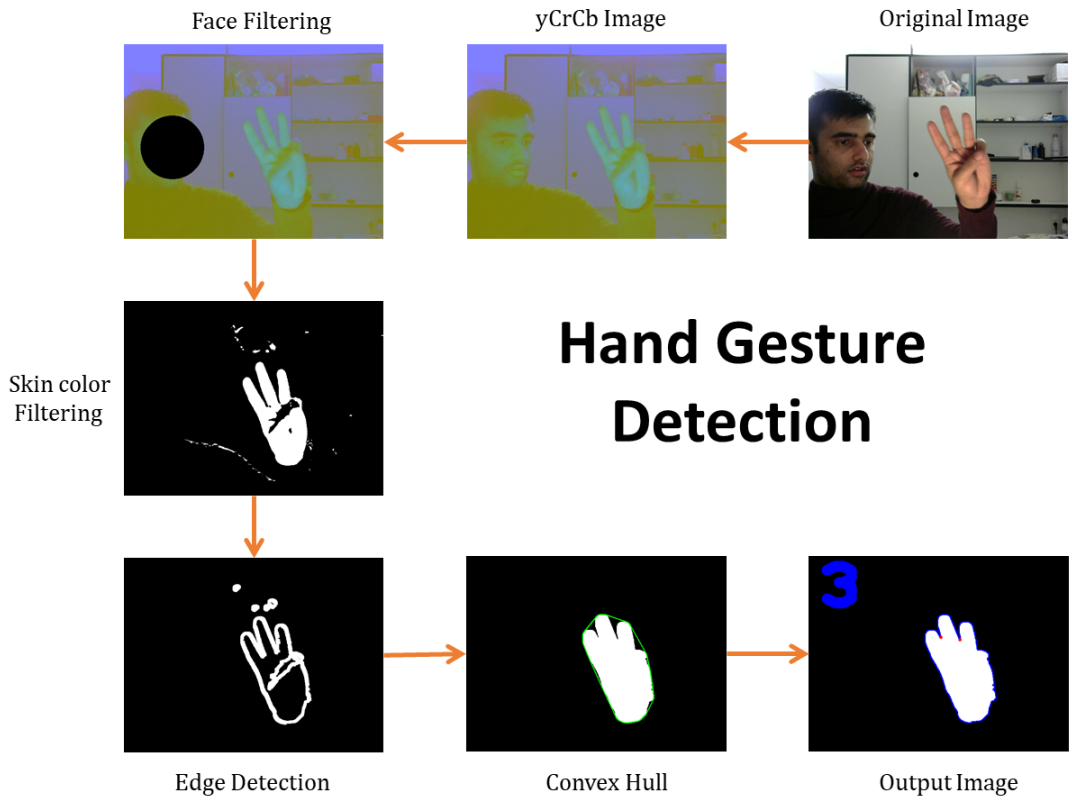


Figure 3.9: Overall pipeline for Hand Gesture Detection.

3.3.2 MediaPipe

The MediaPipe hands first performs a pose estimation of the hand and locates the palm point. It locates all of the finger joints that branch out from the palm point. Figure (3.10) shows an example of MediaPipe hand pose estimation. MediaPipe offers a solution API that can be used from hand gesture detection. It is available for various platforms. This project was done using the solution API for Python. It effectively manages resources by utilizing the CPU or GPU.

Figure 3.10: Hand pose estimation of MediaPipe.

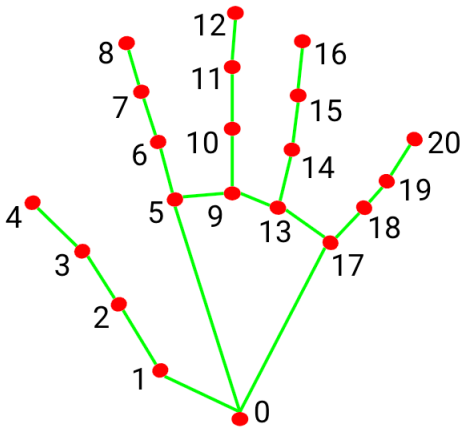
There are various configuration settings that needs to be applied and the configuration that was used for the detections are discussed below.

- STATIC_IMAGE_MODE: This is a flag to decide which mode is used for reading the image. Default is set to false. In this project, the data set was created beforehand so this flag was set to false. As same data set needs to be used to compare with the classical model.
- MAX_NUM_HANDS: This is an integer variable that needs to be defined based on how many hands are being tested. The default value is 2 but for this project it was set to 1.
- MODEl_COMPLEXITY: This is a flag that is used to get better accuracy. If left at the default value of 1 the model is assumed to be complex and accuracy is improved compared to when the value is changed to 0. There is a tradeoff between accuracy and latency. The tradeoff needs to be taken into consideration while using the flag. Changing the value to 0 decreases the latency but accuracy will also decrease. For this project, default value of 1 was used.
- MIN_DETECTION_CONFIDENCE: This is the confidence value that the hand detection model uses to threshold the output. If the confidence is below this given threshold value, then the detection is considered unsuccessful and no output is given. The value must be in the range of $[0.0, 1.0]$. The default value of 0.5 was used for this study.
- MIN_TRACKING_CONFIDENCE: This flag is used for live mode and ignored for the static mode. This value must be in the range of $[0.0, 1.0]$ and used during live hand tracking. This is used to make sure that the hand is tracked successfully. The

default value of 0.5 is used. If the confidence is below the given value, it simply skips the detection to the next image when live tracking. For static mode, every image is used for hand detection [17].

We obtained one 3D landmarks of the hand from the MediaPipe API, as well as hand presence (whether a hand is present in the image or not) and handedness (which hand is present in the image). As this study was concerned with counting the elevated fingers, the co-ordinates of the finger tips were compared to other joint co-ordinates in the finger and this was used to create a simple rule classifier. The steps for this are given below.

- MULTI_HAND_LANDMARKS: This is the collection of detected hands. For this project, only one hand was tested so gave us the list of 21 hand landmarks in world co-ordinates. Each landmark consisted of x, y and z co-ordinates where x and y were normalized to [0.0, 1.0] by the image width and height respectively and z was the depth with the origin at wrist. This means that the landmark is closer to the camera when the z value is smaller .



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 3.11: Hand landmark given my MediaPipe, Figure from [17]

- MULTI_HAND_WORLD_LANDMARKS: This is similar to MULTI_HAND_LANDMARKS but the x, y and z co-ordinates are real world co-ordinates in meters with the origin at hands geometric center.

- MULTI_HANDEDNESS: To determine which hand was present in the image, the Handedness flag was used. If the image contains multiple hands, we'll get a collection of hand landmarks in world coordinates. However, only one hand is relevant for this project. The hand is made up of two parts: a label that is the string "Left" or "Right," and a score that is the probability of the predicted handedness, which must always be greater than 0.5. It's also worth noting that MediaPipe automatically flips the image when it detects that it's being accessed via the webcam or the front

camera. If the other camera is used,the image should be flipped to determine correct handedness [17].

Based on the above information, a simple rule classifier was created to determine the gesture. For example, to decide whether the index finger is elevated, 8^{th} landmark's y co-ordinate can be compared to other landmark in the same finger such as landmark number 7,6, or 5 as given in 3.11. Following the normal image convention, the pixel at (0,0) is the top left and the x co-ordinate increases as we go to the right while y co-ordinated increases as we go down. If the index finger is elevated then y value of 8^{th} landmark will less than the y value of other landmarks(7,6,5) and if the index finger is closed then the y value will be more as it will be below the other landmark of the same finger. This will be true for other fingers except for the thumb. For the thumb, the x value of the 4^{th} landmark needs to be checked as y value can be less even when the finger is closed. X value for tip of the thumb will be less when it is opened and it will be more when the finger will be closed. To make it rotationally invariant, wrist point (0^{th} landmark) can be checked, as it should always have the maximum y value. In addition, x co-ordinate can also be checked and it should be made sure that wrist point doesn't have the minimum x or the maximum x co-ordinate as this could be the case when it is horizontally flipped. But for most cases checking the y co-ordinate should suffice.

3.4 Experiment and Results

Initially, the data was collected for different gestures, i.e. in our problem statement, images for hand gestures with 0, 1, 2, 3, 4, and 5 fingers were taken via webcam. After that, all those images were kept in their individual folder. During data processing, the images were mixed in a single folder, which then were read by the two approaches; classical and MediaPipe.

For both approaches, the experiment setup was done as discussed in subsection (3.3.1). The images were read from a folder with all the hand gestures. While comparing a single gesture, say 3, the rest of the gestures 0, 1, 2, 4, and 5 acted as the negative image. This is the OVR method. The calculation would result in metrics such as accuracy, precision, recall and F1 score.

3.4.1 Classical Approach

The metric calculation for classical approach is summarized in Table (3.5). The output image is shown in Figure (3.13). Computation of all the metrics are given in Table (3.4). The accuracy for classical approach is 0.885. The accuracy of the classical technique showed that it is accurate at recognizing gestures. However, this was not the case. As there were five hundred true positive images and two thousand five hundred true negative images,

the data between true positives and true negatives are imbalanced and accuracy becomes a misleading metric to use.

Table 3.5: Evaluation metrics for Classical approach.

	Class	Precision	Recall	Accuracy	F1-score
	0	0.50	0.82	0.83	0.62
	1	0.47	0.77	0.82	0.59
	2	0.67	0.80	0.90	0.73
	3	0.67	0.74	0.90	0.70
	4	0.75	0.75	0.92	0.75
	5	0.87	0.74	0.94	0.80
Average		0.66	0.77	0.885	0.70

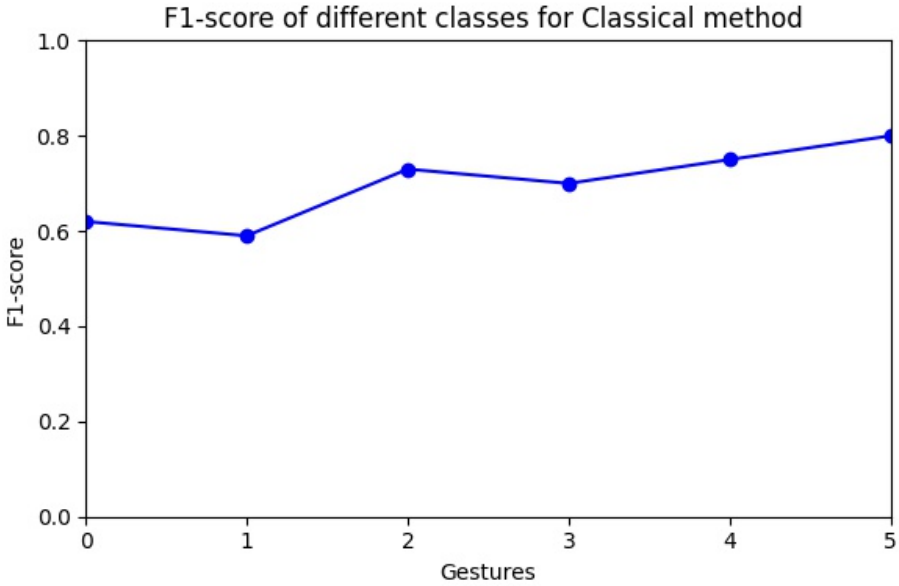


Figure 3.12: F1 score of all the classes for Classical approach.

The large number of true negative samples skews the accuracy metric and suggests that our gesture recognition is better than what it actually is. This does not tell us how accurately the positive samples are detected by our gesture recognition system. For this, we can look at other metrics such as precision and recall. Precision for our classical approach was only 0.66 while the recall was 0.77. It suggests that our gesture recognition is not as accurate as suggested by the accuracy metric. Precision is the number of true positives, out of all the positively predicted samples, while recall is the number of true positives, out of

all the actual positive samples. These are good metrics to use for this study as class imbalance between the low number of true positive samples and high number of true negative samples do not have much effect.

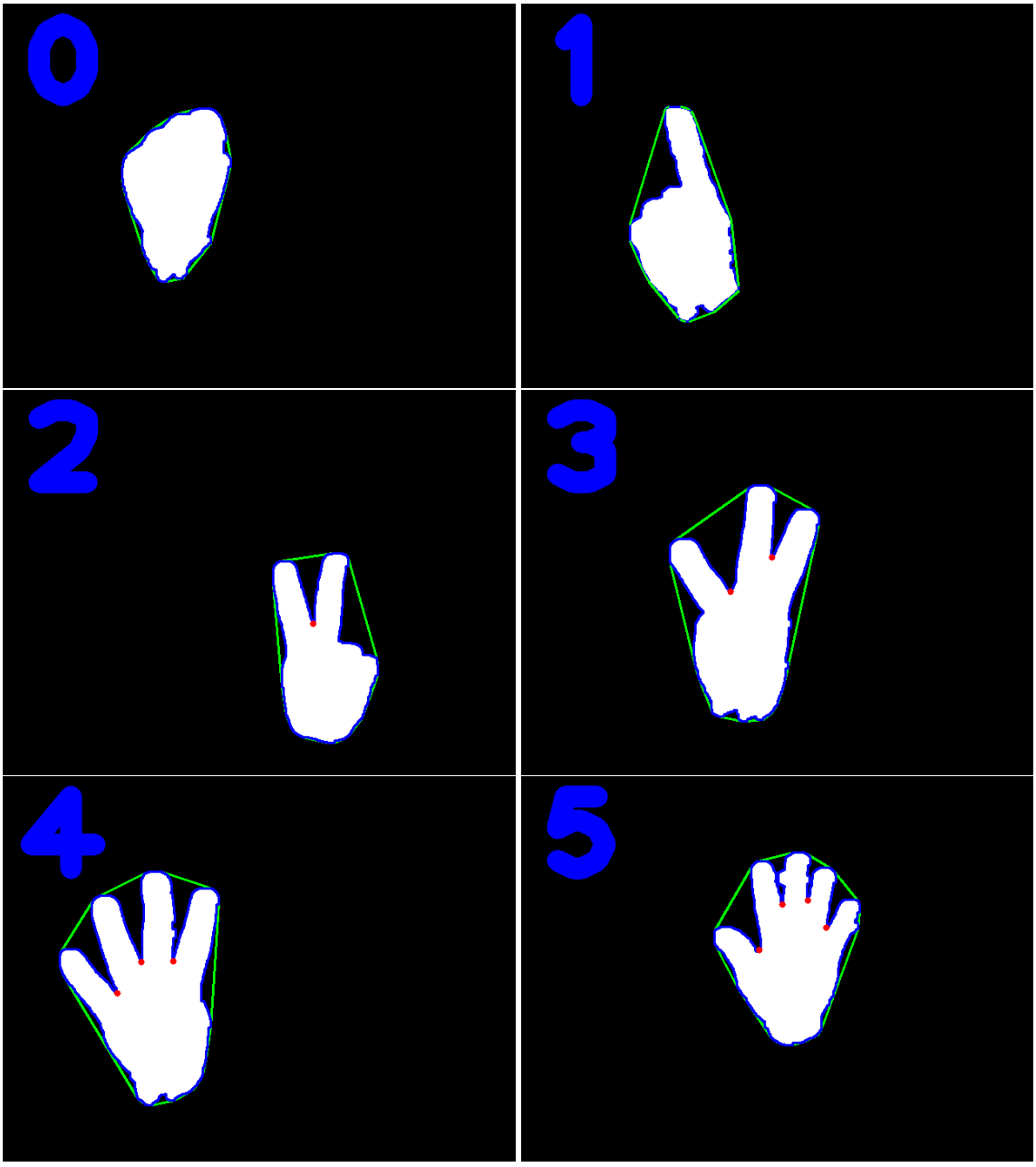


Figure 3.13: Results of Classical approach on the left and results of MediaPipe on the right.

There is a trade off between precision and recall as increasing precision leads to decreasing recall and vice versa. Therefore, F1 score was used as it takes into account both of the metrics. F1 score for our classical example was 0.70 which is not as good as the accuracy metric had suggested. F1-score of all the classes are given in Figure (3.12). Class 5 had the greatest F1-score while class 1 has the lowest F1-score. This is because convexity defect points are easier to detect for 5 fingers because of large gaps in between the fingers while for class 1 only the finger tip detection was used as no convexity defect is detected for this class.

In figure (3.13), the green polygon around the hand represents the convex hull while the red dots represents the convexity points. The contour of the hand for 0 is circular, hence there are no convexity defects and thus no convexity point. For 1, the contour consists of 1 finger tips and no convexity defect as the defect point. The count for the rest of the fingers can be calculated by using the number of convexity defect found. The figure with different points and regions of interest can be shown in Figure (3.14).

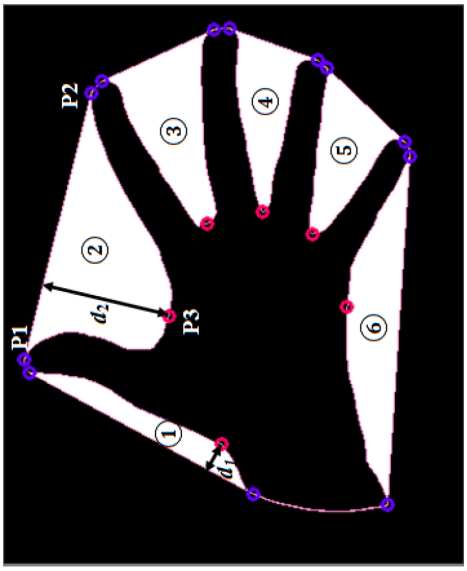


Figure 3.14: Convex Hull, finger tips and convexity points, Figure from [34].

3.4.2 MediaPipe

The metric calculation for MediaPipe is given in 3.6 and examples of results are given in 3.16.

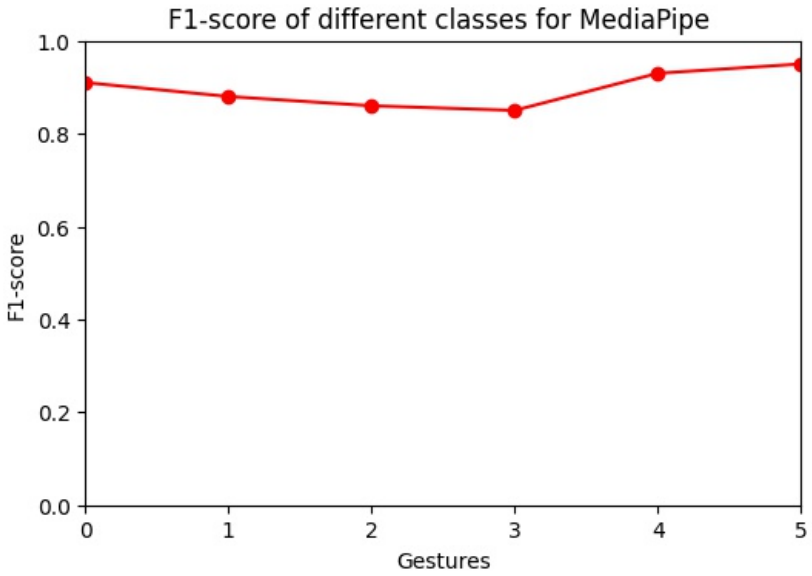


Figure 3.15: F1 score of all the classes for MediaPipe.

Table 3.6: MediaPipe Results.

	Class	Precision	Recall	Accuracy	F1-score
	0	0.99	0.84	0.97	0.91
	1	0.83	0.93	0.96	0.88
	2	0.77	0.98	0.95	0.86
	3	0.97	0.76	0.96	0.85
	4	0.99	0.88	0.98	0.93
	5	0.90	1	0.98	0.95
Average		0.91	0.90	0.97	0.90

MediaPipe had an accuracy of 0.97. This metric was high but this also suffers from the same imbalanced data set as with the classical approach. The precision and recall for MediaPipe is 0.91 and 0.90. The F1 score likewise is 0.90.

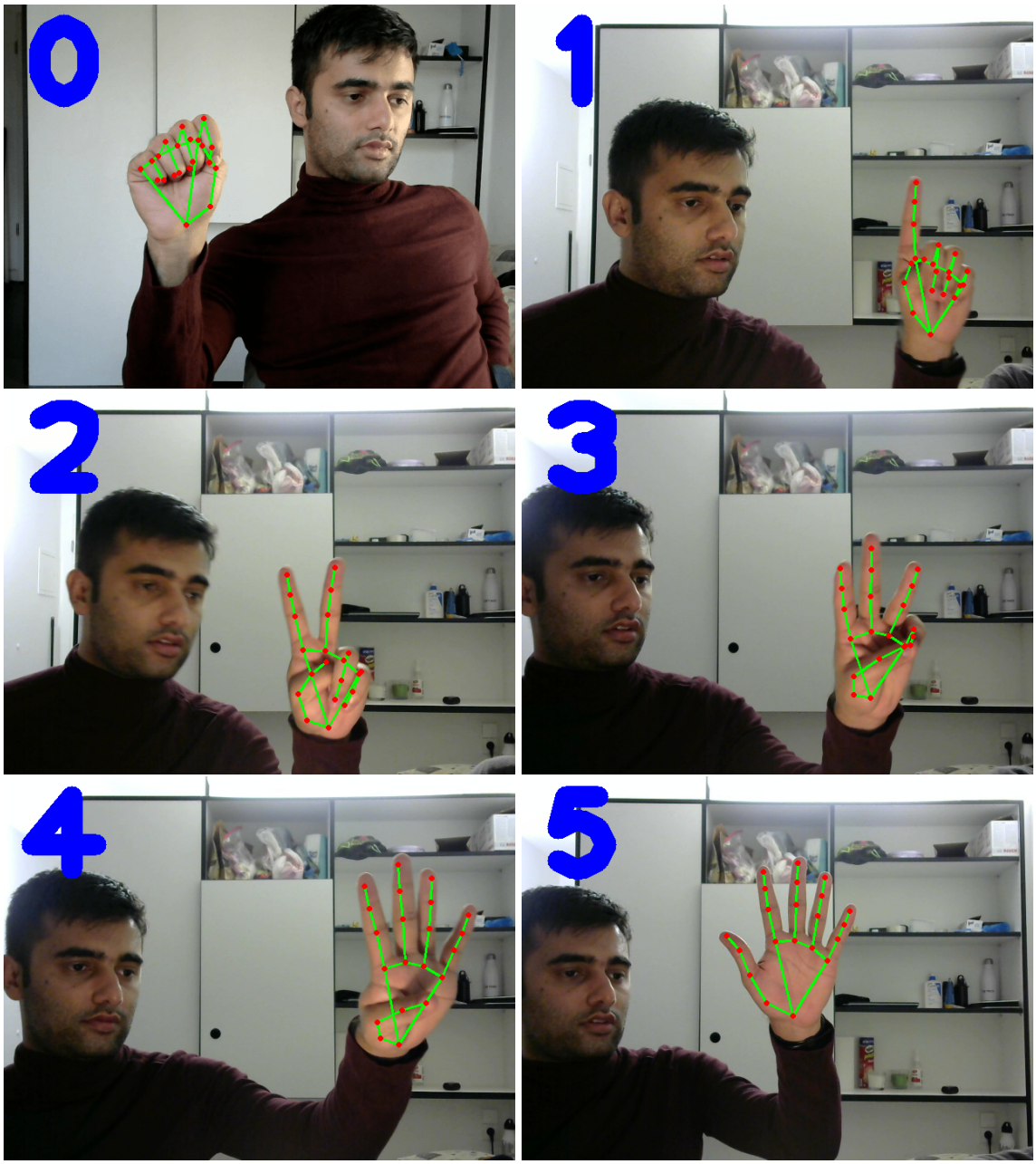


Figure 3.16: Results of MediaPipe hand pose estimation and gesture recognition.

For MediaPipe, ROC curve was also computed. The detection threshhold configuration for MediaPipe was changed from 0.1 to 0.9 with a step size of 0.9 and TPR and FPR was computed. The ROC is curve is given in Figure (3.17).

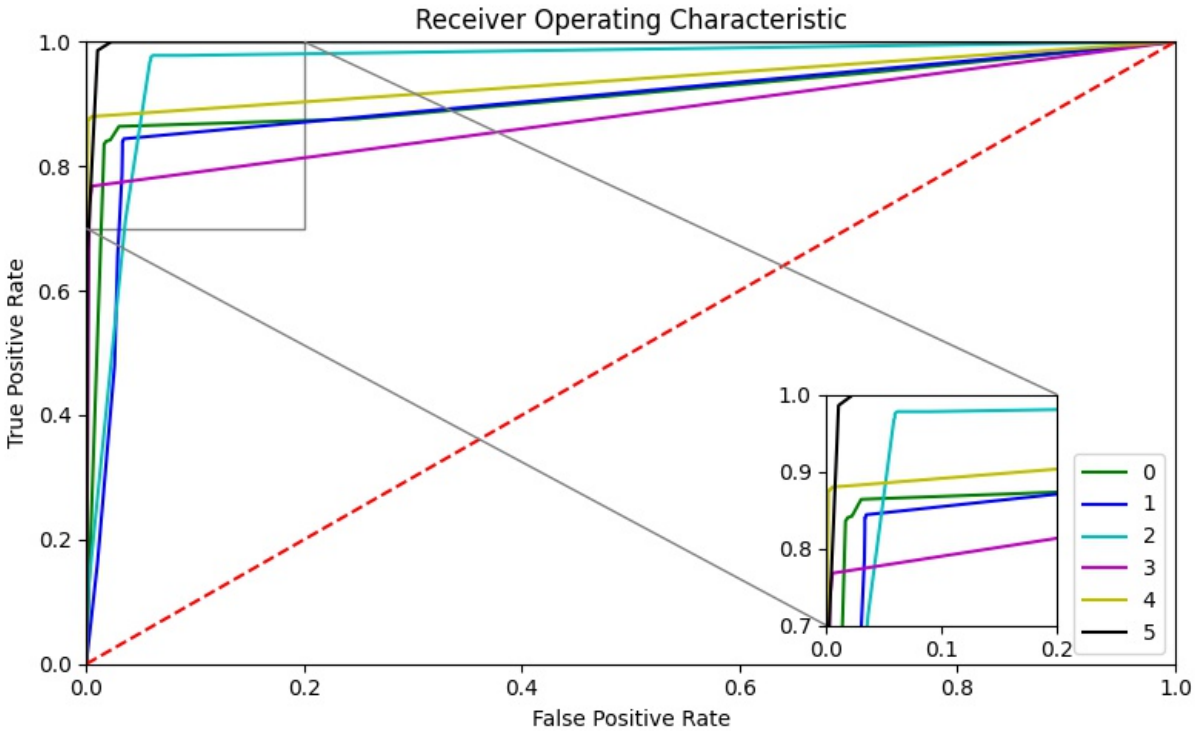


Figure 3.17: ROC curve for MediaPipe.

ROC curves helps us validate if the classifier works better than the random classifier. The random classifier is denoted by the red line. Any curve below will mean that FPR is higher than the TPR. This means that it performs worse than any random classifier. If the curves are all above like in our case in Figure (3.17), it means that our classifier works better than the random classifier. The better classifier will go towards the top left corner. We can see that this is the case for class label 5 with five elevated fingers. However, class label 3 with 3 elevated fingers, performed the worse. Using the ROC curve, the detection threshold of 0.5 was used as it maximized TPR and minimized FPR.

3.4.3 Comparative Analysis of Classical and MediaPipe Approach

The F1-score for all classes in MediaPipe is greater than for the classical approach as shown in Figure (3.18). This showed that MediaPipe approach for gesture detection is much more reliable than the classical approach. MediaPipe had an average F1-score of 0.90 compared to F1-score of only 0.70 for classical approach.

Not only that, but the traditional method took a lot longer than MediaPipe. Table (3.7)

shows the average time taken to run the gesture recognition. In comparison to the Classical technique, MediaPipe was 3.28 times faster. This is comprehensible because MediaPipe has already been trained and only needs to process the images to recognize hand gestures, whereas the traditional technique requires each image to run all of the algorithms from skin filtering through to convexity defect detection. This reduced latency is also due to MediaPipe's efficient resource management (CPU and GPU).

Table 3.7: Average time taken per class for hand gesture detection.

	MediaPipe	Classical approach
Average time (seconds)	86.11	329.06

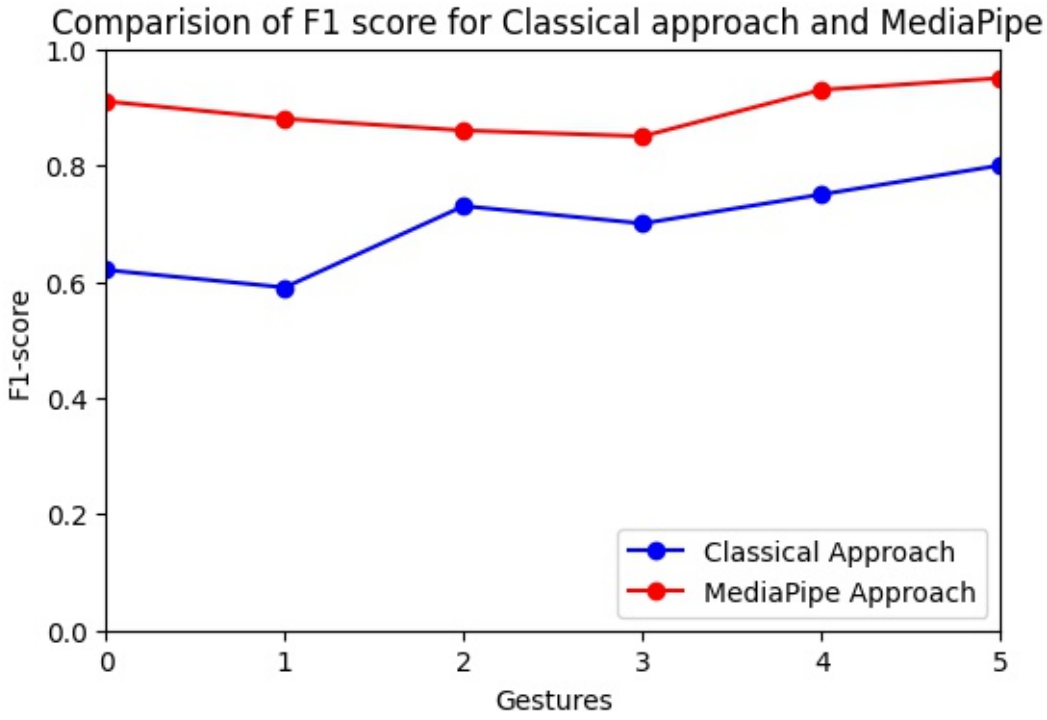


Figure 3.18: Comparison of F1-score for Classical approach and MediaPipe.

From the Figure (3.18), we can observe the comparison of the F1 score between classical approach and MediaPipe. It is clear from the figure that the MediaPipe can recognize hand gesture better than the classical approach.

The classical approach, on the other hand, is simple to implement and adapt to the

needs of the task. It does not require a large amount of data to train and is capable of detecting hand gestures relatively well. As a result, the classical method is inexpensive and can be used for gesture recognition with minimal hardware. For MediaPipe, hundreds of thousands of data points from various sources must be trained. When widely used, there is a risk of bias towards the data set on which the model was trained, which can have a significant impact on the experiments and our society. If the model needs to be re-trained, the cost can be quite high.

4 Conclusions

4.1 Summary

In this project, two different hand gesture detection methods (Classical approach and MediaPipe Hands) were compared by trying to detect six simple gestures. Both methods were used to calculate the number of elevated fingers in the image. Classical approach uses the combination of algorithms from computer vision for the task while MediaPipe Hands is a solution offered by Google. Data set consisted of six classes with five hundred images each. Then, the two different methods were used to detect hand gestures from those static images. OVR was used to binarize the problem for the evaluation. Classical approach had lower F1-score for all classes compared to Mediapipe. Classical approach is only limited to the six simple gestures that was tested on, while MedciaPipe can detect much more complex gestures.

4.2 Discussion

This study compared two different approaches to hand gesture detection. The classical approach using computer vision algorithms and new convolutional neural network solution offered by Google. Several classical approaches that have been discussed have largely been missing the other body parts in the images used for detection, for example in [24], [3]. The images that have been used in those paper only contain the hand. But with the approach that has been discussed in this project, the hand gesture can be detected even when other skin pixels such as the face is in the image. Our approach can detect hand gesture sufficiently well. However, it is not as good as the solution offered by MediaPipe. Our approach can only detect simple gestures such as counting the elevated fingers as there's no clear distinction made between which fingers are raised. This approach can be used for simple task and it is limited to detect six gestures.

The dataset consisted of five hundred images in each class. The multiclass classification was changed to binary classification for the evaluation. OVR method was used to binarize the problem which resulted in each class consisting of five hundred true positive image and two thousand five hundred negative images. This made the dataset imbalanced. Therefore, F1 score was chosen to evaluate the two approaches as accuracy would have been skewed due to the large number of negative samples.

For the classical approach yCrCb colour space was chosen for the skin colour filtering. This process was also tried with HSV colour space but the skin detection was not as accurate. The example of skin detection and face removal using HSV model is given in Figure (4.1). The other challenges included finding the contour while the hand occluded some part the face. This meant that the face detection failed and the contour that was found was not only of the hand but also of the face. This often resulted in incorrect gesture detection.

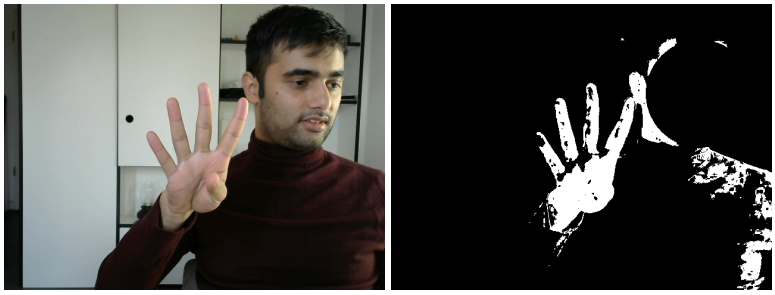


Figure 4.1: Results of skin colour filtering after face removal for HSV colour space (original image in the left and skin colour filtering HSV colour space on the right)

For, Mediapipe the detection thresh hold value chosen was 0.5 as it minimized FPR while maximizing the TPR. Only one hand was used for the detection task, so that it could be compared with the classical approach.

As we can see in Table (3.5), the average F1-score was 0.70 for the classical approach. But, for MediaPipe the average F1-score was 0.90 given the same dataset. This means that MediaPipe is much more reliable detection system for gesture recognition. Not only this but the Mediapipe can distinguish between specific fingers that are detected and can be used to solve more complicated tasks such as to detect sign language.

4.3 Outlook

There are other related concerns that need to be addressed in the future, particularly for the classical approach. If there is no hand in the image and a random item with a similar tone to the skin pixel is discovered in the background, the maximum contour of that region is found and the convex hull of that area is computed. Furthermore, if the hand is inserted after that, the contour region is added, and convexity points are produced for both the hand and the background region. This will need to be addressed in the future for a more reliable detection. This is because the skin color filtering method used to detect the skin pixels isn't totally correct. New gesture control technologies, such as MediaPipe, are

a much more reliable and efficient choice for gesture detection tasks.

Even for MediaPipe, there is a however a limitation with the laptop's web cameras as they are not as good at the moment. Due to the Covid 19 pandemic, lots of laptop manufacturing companies have started putting better cameras as webcams has been a necessity for working from home. For this project, older laptop camera sometimes struggled due the noise but once Logitech C270 camera was used, the images were smoother and this significantly improved hand gesture recognition. The selfie camera of smart phones are also getting much better which can drastically improve the detection.

MediaPipe's hand gesture recognition system shows real promise and can be reliably used to create applications that relies on gesture detection. Mediapipe could be further integrated into other apps such as google maps to navigate the map without touching the screen.

List of Figures

1.1 Summary overall pipeline for Hand Gesture Detection. 3

2.1 Output of skin colour filtering, Figure from [24]. 7
2.2 City Block Distance transform on (a) binary image (b) distance transform, Figure from [3]. 10
2.3 Results after finding the palm point, inner circle with maximal radius, wrist points, and the wrist lines, Figure from [3]. 11
2.4 Yellow line is the palm line and red line parallels the wrist line, Figure from [3]. 11
2.5 Convex set in the left and non-convex set in the right, Figure from [2]. . . . 13
2.6 Convexity defects and the depths d_i, Figure from [32]. 13
2.7 Single Shot Detector (SSD) framework, loc is location of the bounding box, conf is the confidence of all object categories, Figure from [16]. 16
2.8 Feature Pyramid Network (FPN), Figure from [14]. 17
2.9 Example of hand pose estimation using MediaPipe, figure from [35]. 18
2.10 Architecture of hand landmark model for MediaPipe, figure from [35]. . . . 19

3.1 Example of Input Image data for 6 classes. 22
3.2 Example of an ROC curve where the red line indicates how a random classifier would work, blue dot represent the perfect classifier while the cyan is the ROC curve. 26
3.3 Original image on the top, image in yCrCb colour space on the middle and image after applying skin colour filtering on the bottom. 27
3.4 The output after applying face detection and cutting out the face pixel on the yCrCb image (Left). The output after applying skin filtering on the left image (Right). 28
3.5 Applying sobel operator to binary image after skin color detection. 29
3.6 Convex hull of an image. 30
3.7 Convexity defect. 31
3.8 convexity defect with angle γ. 31
3.9 Overall pipeline for Hand Gesture Detection. 32
3.10 Hand pose estimation of MediaPipe. 33
3.11 Hand landmark given my MediaPipe, Figure from [17] 34
3.12 F1 score of all the classes for Classical approach. 36
3.13 Results of Classical approach on the left and results of MediaPipe on the right. 37

3.14 Convex Hull, finger tips and convexity points, Figure from [34]. 38
3.15 F1 score of all the classes for MediaPipe. 39
3.16 Results of MediaPipe hand pose estimation and gesture recognition. 40
3.17 ROC curve for MediaPipe. 41
3.18 Comparison of F1-score for Classical approach and MediaPipe. 42

4.1 Results of skin colour filtering after face removal for HSV colour space (original image in the left and skin colour filtering HSV colour space on the right) 45

List of Tables

2.1 Hand gesture recognition system classification, Table from [12]. 4
2.2 Robert (G_x) 9
2.3 Robert (G_y) 9
2.4 Sobel (S_x) 9
2.5 Sobel (S_y) 9
2.6 Perwitt (P_x) 9
2.7 Perwitt (P_y) 9

3.1 Classifiers for one vs rest. 23
3.2 Confusion matrix 24
3.3 Example for class 3, which can be interpolated to other classes. 24
3.4 Metrics used in the evaluation. 24
3.5 Evaluation metrics for Classical approach. 36
3.6 MediaPipe Results. 39
3.7 Average time taken per class for hand gesture detection. 42

Glossary

FPN Feature Pyramid Network. 15–17, 47

FPR False Positive Rate. 25

GT Ground Truth. 15

HCI Human Computer Interaction. 1

HSV Hue, Saturation, Value. 6

mAP mean Average Precision. 15

RCNN Regional Convolutional Neural Network. 15

RGB Red, Green, Blue. 5

ROC Receiver Operating Characteristics. 25, 40

SE structuring element. 8

SSD Single Shot Detector. 4, 15, 16, 47

TPR True Positive Rate. 25

YOLO You Only Look Once. 15

Bibliography

[1] Hani Al-Mohair, Junita Mohamad-Saleh, and Shahrel Azmin Suandi. Color space selection for human skin detection using color-texture features and neural networks. In *Color space selection for human skin detection using color-texture features and neural networks*, pages 1–6, 06 2014.

[2] Djordje Baralic and Lazar Milenkovic. Surprising examples of manifolds in toric topology!, 2017.

[3] Zhi-Hua Chen, Jung-Tae Kim, Jianning Liang, Jing Zhang, and Yu-Bo Yuan. Real-time hand gesture recognition using finger segmentation. *TheScientificWorldJournal*, 2014:267872, 06 2014.

[4] Ananya Choudhury, Anjan Talukdar, and Kandarpa Sarma. A novel hand segmentation method for multiple-hand gesture recognition system under complex background. In *A Novel Hand Segmentation Method for Multiple-Hand Gesture Recognition System under Complex Background*, 02 2014.

[5] Open CV. Haar cascades. https://github.com/opencv/opencv/tree/master/data/haarcascades, 2020. accessed: 30/10/2021.

[6] Qing Gao, Yongquan Chen, Zhaojie Ju, and Yi Liang. Dynamic hand gesture recognition based on 3d hand pose estimation for human-robot interaction. *IEEE Sensors Journal*, PP:1–1, 05 2021.

[7] Giovani Gomez and Eduardo Morales. Automatic feature construction and a simple rule induction algorithm for skin detection, 07 2002.

[8] Ivan Grishchenko and Valentin Bazarevsky. Mediapipe holistic - simultaneous face, hand and pose prediction, on device, Dec 2020.

[9] Yixuan He, Tianyi Hu, and Delu Zeng. Scan-flood fill(scaff): an efficient automatic precise region filling algorithm for complicated regions. *CoRR*, abs/1906.03366, 2019.

[10] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus, 07 2019.

[11] Harpreet Kaur and Jyoti Rani. A review: Study of various techniques of hand gesture recognition. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pages 1–5, 2016.

[12] Rafiqul Zaman Khan and Noor Ibraheem. Survey on gesture recognition for hand image postures. *Computer and Information Science*, 5, 04 2012.

[13] Yang LI, Jin HUANG, Feng TIAN, Hong-An WANG, and Guo-Zhong DAI. Gesture interaction in virtual reality. *Virtual Reality & Intelligent Hardware*, 1(1):84–112, 2019.

[14] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Feature Pyramid Networks for Object Detection*, pages 936–944, 07 2017.

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Focal Loss for Dense Object Detection*, pages 2999–3007, 10 2017.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. Ssd: Single shot multibox detector. In *SSD: Single Shot MultiBox Detector*, volume 9905, pages 21–37, 10 2016.

[17] Google LLC. Hands. https://google.github.io/mediapipe/solutions/hands.html. accessed: 01/11/2021.

[18] Ognjan Luzanin and Miroslav Plancak. Hand gesture recognition using low-budget data glove and cluster-trained probabilistic neural network. *Assembly Automation*, 34, 01 2014.

[19] Khairul Anuar Mat Said, Asral Jambek, and Nasri Sulaiman. A study of image processing using morphological opening and closing processes. *International Journal of Control Theory and Applications*, 9:15–21, 01 2016.

[20] Naima Otberdout, Lahoucine Ballihi, and Driss Aboutajdine. Hand pose estimation based on deep learning depth map for hand gesture recognition. In *2017 Intelligent Systems and Computer Vision (ISCV)*, pages 1–8, 2017.

[21] Munir Oudah, Ali Abdulelah Al-Naji, and Javaan Chahl. Hand gesture recognition based on computer vision: A review of techniques. *Journal of Imaging*, 6:73, 07 2020.

[22] Prashan Premaratne. *Human Computer Interaction Using Hand Gestures*. Springer Singapore, 2014.

[23] Romi Rahmat, Tengku Chairunnisa, Dani Gunawan, and Opim Sitompul. Skin color segmentation using multi-color space threshold. In *Skin Color Segmentation Using Multi-Color Space Threshold*, 08 2016.

[24] D.J. Rios-Soria, S.E. Schaeffer, and S.E. Garza-Villarreal. Hand-gesture recognition using computer-vision techniques, 01 2013.

[25] Atsushi Shimada, Takayoshi Yamashita, and Rin-ichiro Taniguchi. Hand gesture based tv control system — towards both user- amp; machine-friendly gesture applications. In *The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, pages 121–126, 2013.

[26] G.T. Shrivakshan and Chandramouli Chandrasekar. A comparison of various edge detection techniques used in image processing. *International Journal of Computer Science Issues*, 9:269–276, 09 2012.

[27] Ade Silvia and Nyayu Latifah Husni. Hand contour recognition in language signs codes using shape based hand gestures methods. In *Hand Contour Recognition in Language Signs Codes Using Shape Based Hand Gestures Methods*. Sriwijaya University, 1st International Conference on Computer Science and Engineering, 9 2014.

[28] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Hand Keypoint Detection in Single Images Using Multiview Bootstrapping*, pages 4645–4653, 07 2017.

[29] Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith, and Jon Handler. *A Real-Time Hand Gesture Interface for Medical Visualization Applications*, volume 36, pages 153–162. Springer, 04 2010.

[30] Tilo Strutz. The distance transform and its computation. *CoRR*, abs/2106.03503, 2021.

[31] Bex T. Comprehensive guide to multiclass classification metrics. https://tinyurl.com/zd6emypd. accessed:04/11/2021.

[32] Ghassem Tofighi, Amirhassan Monadjemi, and Nasser Ghasem-Aghaee. Rapid hand posture recognition using adaptive histogram template of skin and hand edge contour. *2010 6th Iranian Conference on Machine Vision and Image Processing, MVIP 2010*, 10 2010.

[33] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[34] Yanan Xu, Dong-Won Park, and Gouchol Pok. Hand gesture recognition based on convex defect detection. 2017.

[35] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 06 2020.