## Project Report On

# AIR CANVAS

**Presented By:**
**B.SHIVANI**
**V.MANYA**

# TABLE OF CONTENTS

11 **LIST OF DIAGRAMS**

# ABSTRACT

In recent years, the fusion of computer vision techniques and interactive technologies has opened up many possibilities for creative expression. This abstract introduces "Air Canvas," which transforms the physical space into a digital canvas where users can paint and draw simply by moving their hands or any other object in the air. With the occurrence of unexpected circumstances and pandemics, there has been a need for virtual reading and learning, our project is a medium through which that need can be fulfilled in an interesting and exploring manner

The project uses the concept of computer vision which is a branch of Artificial Intelligence (AI) that uses various processing techniques, machine learning, and deep learning methods to replicate and automate tasks that require human visual perception. This includes **object detection, recognition,** tracking, scene understanding, **image segmentation**, and more. This includes the use of various algorithms such as **Convolutional Neural Networks (CNNs)** which are commonly used for detecting and localizing objects within images, including hands or specific objects in the camera's field of view

These algorithms can be implemented using various Machine learning libraries and frameworks in Python, such as OpenCV. By integrating these algorithms into the Air Canvas project, the application can offer a more immersive and intuitive painting experience for users, enhancing creativity, learning, and interaction.



**Keywords**: Object recognition, CNN, image segmentation

**Domain**: Machine learning, Image Processing

# 1. INTRODUCTION

## 1.1 Introduction to Project

With the growing Digitalization , we have seen everything to be available in a digital mode..from Shopping , Booking , Planning , E-commercing , Reading ,Dropshipping, Streaming , Blogging , Payments to now where we have achieved the need of online education and teaching. This has facilitated the introduction of various educational platforms as well it has led to a creative atmosphere of user interacting services which can be accessed easily throughout the world.

As a result , the search for such interactive systems and atmosphere is ongoing . An attractive visualization of such a service can be a platform where people of all age groups can interact and elaborate their creativity by drawing things in the open air and creating a visualization of what can be drawn in the air. Such a technology can be a step forward to a vision where we could actually be able to move things in air and generate high tech images and services with its help.

The Air Canvas Project aims to provide a unique , contactless drawing experience using webcam and sophisticated hand tracking mechanism and algorithms , using which users can draw , select colors and clear the canvas by simply moving on to the camera.

The main Technologies used in this project includes the following :

- OpenCV
- Mediapipe
- Numpy
- Python
- Collections (Deque)

Air canvas is hence an interactive innovative application used with the collaboration of computer vision and machine learning technologies to create an interface using various technologies and algorithms.

This project can be divided into multiple crucial phases:

- Video capture: It is the process of gathering frames from the camera.
- Hand Detection and Tracking: This technique uses machine learning techniques to identify and monitor hand movements.
- Hand gesture recognition: identifying particular hand motions to direct the actions of the drawing.
- Sketching on Canvas: Producing the illustrations by utilizing the identified hand gestures and movements.

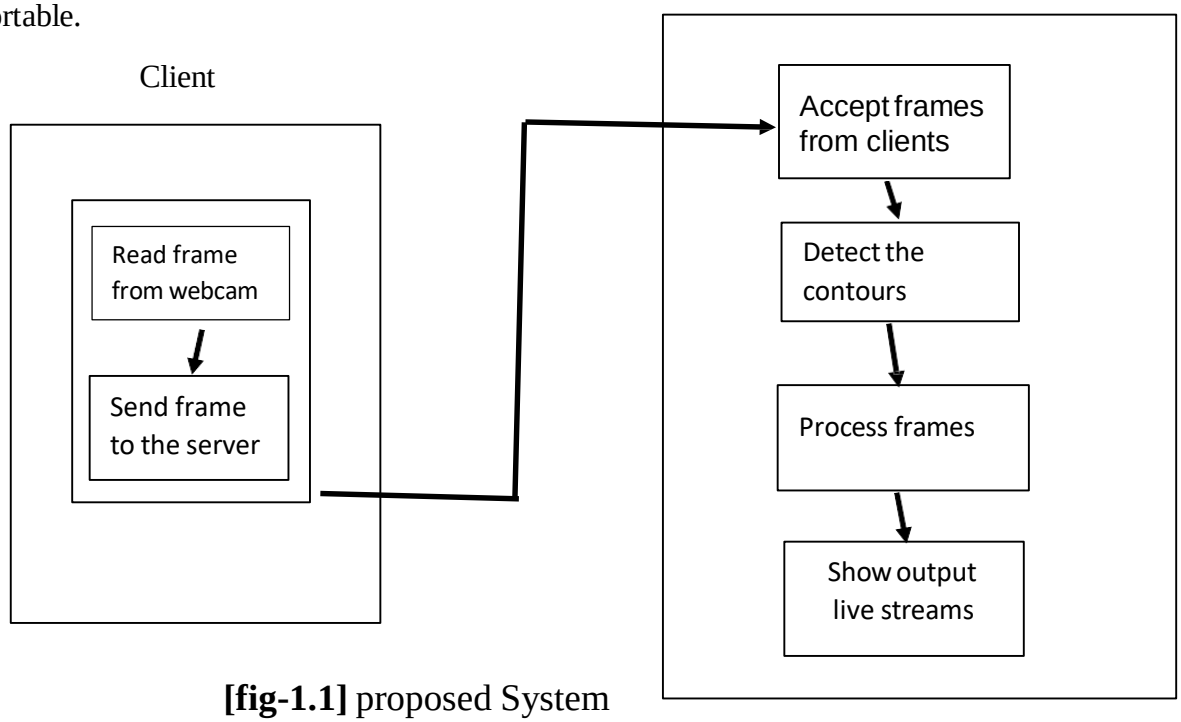
## 1.2 Existing System

An existing system similar to this project is the Google Jamboard application , which is a paid version of a virtual collaboration by google. As a member of the G Suite software family, Google created Google Jamboard, an interactive digital whiteboard. It is intended to support group ideation, group presentations, and group instruction.


**[fig-1]** Jam board

## 1.3 Proposed System

The proposed application is a an open source application can be used free of cost and doesn't require complicated steps or digital equipments for its working and functioning.Although Google Jamboard is an effective tool for teamwork, there are a number of special benefits to the Air Canvas project, especially when it comes to its creative approach to painting and interaction.Which includes Free hands interaction , Open source environment , compatibility and portable.




**[fig-1.1]** proposed System

# 2. REQUIREMENT ENGINEERING

## 2.1 Hardware Requirements

- Any System with the provision of a webcam
- A tool for interaction like Keyboard and Mouse

## 2.2 Software Requirements

- Jupyter Notebook
- Python
- Python Libraries
    1. pandas
    2. numpy
    3. MediaPipe

# 3. LITERATURE SURVEY

Mouse detection with OpenCV and NumPy in Python entails capturing and responding to mouse events such as clicks, motions, and drags in an OpenCV window. OpenCV's setMouseCallback function provides a simple way to handle these events. Here's an explanation and example of how to do this:

OpenCV is used to construct windows and track mouse movements. It supports a variety of mouse event types, including EVENT_LBUTTONDOWN and EVENT_MOUSEMOVE. This function specifies what actions to do when a mouse event happens. The function records the coordinates of the mouse event and can be used to draw on an image or annotate points. An OpenCV window is constructed to show either an image or a blank canvas.This window has a mouse callback function connected. The callback function handles various mouse events using conditional statements to conduct certain actions based on the event type [1]

PAINT WITH MUSIC : An interactive program that creates digital art using hand motions and music. Users can paint on a virtual canvas by tracking their hand movements, and each stroke generates a musical note, resulting in a full multimedia experience. Paint with Music is a unique tool that combines visual art and music with hand motions. The device uses a camera or motion sensor to follow the user's hand movements, allowing them to paint on a virtual canvas. Each brush stroke generates a musical note or sound, resulting in a harmonic combination of visual and audio art. This interactive experience allows users to not only create one-of-a-kind digital artworks, but also compose music, making it an appealing tool for artists, musicians, and educators. The initiative stresses the synergy of various kinds of creative expression and increases user engagement through multimodal interaction. [2]

Drag and drop simulation using OpenCV and Python entails designing an interactive graphical interface in which users may click, hold, and drag objects on the screen. This procedure begins with creating an OpenCV window to display a canvas or image. A mouse callback function is defined to handle specific mouse events, such as EVENT_LBUTTONDOWN (representing the beginning of a drag action), EVENT_MOUSEMOVE (indicating an ongoing drag), and EVENT_LBUTTONUP (indicating a drop action). When the left mouse button is pressed down over an object (such as a rectangle), the initial position of the mouse click is saved. As the mouse advances with the button kept down, the object's position is constantly changed to follow the pointer, simulating a dragging motion. When the button is released, the object is 'dropped' to its new location. The display is refreshed in a loop, displaying the object's current position in real time. This configuration allows users to interact with the canvas's visual elements in an intuitive manner, delivering a simple yet effective example of drag-and-drop capabilities [3]

A virtual keyboard built with OpenCV and machine learning techniques creates an interface that allows users to text by interacting with a display via hand gestures or finger movements rather than a traditional keyboard. The system uses OpenCV for real-time video capture and image processing to detect and track hand landmarks and finger locations. Machine learning algorithms, such as those supplied by Mediapipe, are utilized to correctly identify hand gestures and finger touches. The virtual keyboard layout is displayed on the screen, and as the user moves their hand or fingers over the keys, the system interprets their movements to determine the intended key presses. When a finger presses a virtual key, the matching character is entered, simulating typing. This technique provides a frictionless and intuitive typing experience by leveraging computer vision and machine learning capabilities to improve user engagement and accessibility [4]

Volume and video control with OpenCV and machine learning techniques entails using computer vision to recognize hand motions or facial features when interacting with multimedia apps. OpenCV captures video input via a camera and processes the frames to detect specific motions or movements using machine learning models built for tasks such as hand tracking and facial detection. A gesture recognition model, for example, can recognize hand signs that indicate volume up or down, play, pause, or skip commands. When these motions are identified, the system maps them to the proper multimedia controls, allowing users to adjust the volume or manage video playback with simple, touchless interactions. This combination of computer vision and machine learning creates a seamless and effective solution to improve user experience when engaging with digital media.. [5]

The blinking game uses OpenCV and machine learning techniques to create an interactive application that detects eye blinks and controls game features, providing an exciting approach to enhance focus and reaction time. This approach begins with the use of OpenCV for real-time video capture and facial detection. Machine learning techniques, often built on frameworks such as Mediapipe or Dlib, are used to reliably detect and track facial landmarks, with a concentration on the eyes. The algorithm is trained to distinguish blink patterns by examining eye aspect ratios or other metrics. When a blink is detected, a predetermined game action is executed, such as moving a character or selecting an option.The game continuously scans the video feed, adjusting the gameplay based on the user's blinks, resulting in a dynamic and hands-free gaming experience that combines computer vision and machine learning to enable new user engagement. [6]

The Virtual Graffiti Wall uses OpenCV and machine learning to transform the concept of graffiti painting. OpenCV offers real-time video processing and allows the system to recognize and track the user's hand motions or brush strokes with great accuracy. The technology uses computer vision techniques such as object detection and image segmentation to determine the canvas area where users can freely express themselves. Machine learning algorithms play an important role in recognizing specific motions or stances, allowing users to control many features of the virtual spray can, such as color selection, line thickness, and spray intensity. These algorithms learn from

user interactions and constantly improve their accuracy and responsiveness.The virtual environment not only replicates the experience of conventional graffiti, but also improves it with digital elements such as undo options, layering effects, and interactive sharing. This immersive platform promotes artistry, experimentation, and collaboration among educators and enthusiasts, ushering in a new era of interactive digital art experiences.. [7]

The Leap Motion Painting App demonstrates an innovative use of technology for digital art creation. This application uses the Leap Motion Controller to monitor the delicate movements of hands and fingers with high accuracy. This precise tracking capacity allows users to paint and sketch directly on a virtual canvas in real time, transforming small gestures into intricate and detailed artwork. The app's interface features straightforward controls that respond to hand gestures, allowing users to select colors, modify brush sizes, and navigate inside the canvas space. Artists may operate digital tools with fluidity and precision, boosting their creative process and allowing them to experiment with new approaches without the constraints of the physical medium. The Leap Motion Painting App thus combines superior motion tracking technology and artistic expression, allowing users to explore new aspects of digital painting with simplicity and complexity. [8]

# 4. TECHNOLOGY

## 4.1 ABOUT PYTHON

Python is a popular high-level interpreted programming language that is easy to learn and understand. It was developed by Guido van Rossum and published in 1991. It is compatible with procedural, object-oriented, and functional programming paradigms. Python is perfect for web development (Django, Flask), data science (NumPy, pandas), and machine learning (TensorFlow, PyTorch) because of its large standard library and plethora of third-party packages.

Python's sophisticated capabilities attract to seasoned developers, but its simple syntax and intuitive user interface make it accessible to novices as well. Python is widely used in scientific computing, automation, and other fields where it helps with code maintainability and rapid development. Python is still a mainstay of contemporary software development, bolstered by a thriving community and ongoing development.

## 4.2 APPLICATIONS OF PYTHON

Python's flexibility and robust library make it a popular choice across many industries. Frameworks for web development such as Flask and Django make it possible to create reliable web apps. With tools like NumPy, pandas, and Matplotlib for data manipulation and visualization, Python shines in data science and analytics. PyTorch, Keras, and TensorFlow are some of the libraries that improve its machine learning capabilities.

Python is also well-liked for automation, as it makes jobs like system administration and web scraping possible. Libraries such as SciPy and SymPy enable sophisticated simulations and calculations in scientific computing. Furthermore, Python is utilized in the creation of graphical user interfaces using Tkinter and in game development with Pygame. In these varied sectors, Python is the chosen option due to its readability and simplicity.

It is used in various fields:

- Web development
- Game development
- Data analysis
- Automation and scripting
- GUI development
- Machine learning and artificial intelligence

### 4.3.1 NUMPY

Large, multi-dimensional arrays and matrices are supported by NumPy, a core library for scientific computing in Python, which also offers a range of mathematical methods to manipulate the arrays. NumPy, which was developed in 2006, is a popular tool for numerical computations

that makes it possible to do operations like statistical analysis, Fourier transformations, and linear algebra and manipulate arrays efficiently. Its robust N-dimensional array object, nd array, is a crucial tool for applications requiring high performance since it enables effective computation and seamless interaction with C/C++ and Fortran code. NumPy is essential to data science, machine learning, and engineering because it forms the basis for numerous other scientific libraries, including SciPy, pandas, and scikit-learn. A mainstay of Python's scientific computing ecosystem, NumPy is highly functional and intuitive to use.

### 4.3.2 MEDIAPIPE

Google created MediaPipe, a flexible platform for creating multimodal machine learning pipelines that is especially helpful for real-time perception applications. Because it allows for cross-platform deployment, it can be used on desktop computers, mobile devices, and the internet. Pose estimation, object detection, face detection, hand tracking, and other ready-to-use solutions are all available on MediaPipe. Real-time processing even on devices with limited resources is made possible by the framework's utilization of sophisticated machine learning models that are optimized for speed and accuracy. Because of its modular design, developers can expand and alter pipelines to suit certain requirements. MediaPipe is a potent tool for developers working on cutting-edge AI applications because of its interaction with well-known programming environments like Python and C++, which guarantees wide applicability and user-friendliness.

### 4.3.3 OPENCV

An open-source framework for real-time computer vision and image processing is called OpenCV (Open Source Computer Vision Library). It was first created by Intel in 1999 and is compatible with several other programming languages, such as Python, C++, and Java. More than 2,500 well-suited algorithms are available in OpenCV for a range of applications, including motion tracking, picture segmentation, object detection, and facial recognition. It is well-liked in industries like robotics, augmented reality, and machine learning because of its vast functionality and simplicity of usage. OpenCV's capabilities for sophisticated applications are improved by its interaction with deep learning frameworks like as TensorFlow and PyTorch. Developers may create high-performance apps on Windows, Linux, macOS, and mobile platforms thanks to the library's cross-platform support.

### 4.3.4 COLLECTIONS

The `collections` module provides a very effective implementation of a double-ended queue in Python called `collections.deque}. It outperforms lists in situations where quick append and pop operations are needed from both ends of the queue. Important functions are `pop(){ and `popleft()} for element removal and `append(){ and `appendleft()` for adding elements to the right and left ends, respectively. Furthermore, components within the deque can be efficiently rotated with the help of the `rotate()` method. When `deque` is used for activities like scheduling and buffering in different programming contexts, it can be used to restrict the amount of elements by automatically deleting the oldest items when the limit is surpassed. This is possible using the optional `maxlen} parameter.

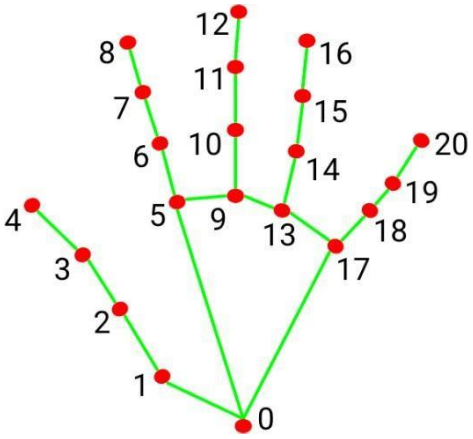
### 4.4.1 Dataset Description

The project uses a pre trained dataset present in the MediaPipe Hand Tracking module that consists of the Hand Tracking landmarks under its solutions.Hands module.

### 4.4.2 Hand Tracking Module

For reliable real-time hand tracking and detection in films and live streams, MediaPipe offers a hand tracking module. It makes use of a machine learning model that has been taught to precisely identify and detect hand landmarks like fingers and joints. Built on top of TensorFlow Lite, the module enables effective inference across a range of platforms, including desktops and mobile devices.

To attain accuracy and real-time performance, the hand identification algorithm integrates computer vision methodology with deep learning techniques. After detecting the existence of hands by processing input frames, it tracks the hands' movements and gestures over time. Because of these features, MediaPipe's hand tracking module is a useful tool for applications that need to recognise sign language, allow hand gestures for control in augmented and virtual reality settings, and allow precise interaction with virtual objects.




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

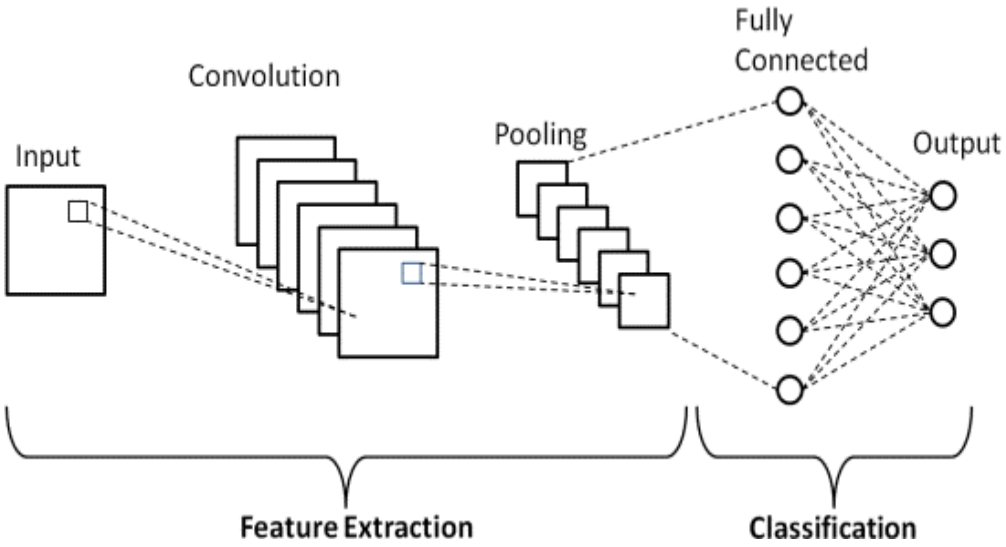
**[fig-2]**Landmarks

## 4.5 CNN

Convolutional neural networks, or CNNs, are a subclass of deep neural networks that are mostly used to interpret visual input, including pictures and movies. Its capacity to automatically derive hierarchical feature representations from pixel data is what makes it unique. Convolutional layers, which apply filters to input data, pooling layers, which downsample features, and fully connected layers, which are used for classification or regression tasks, make up CNNs.

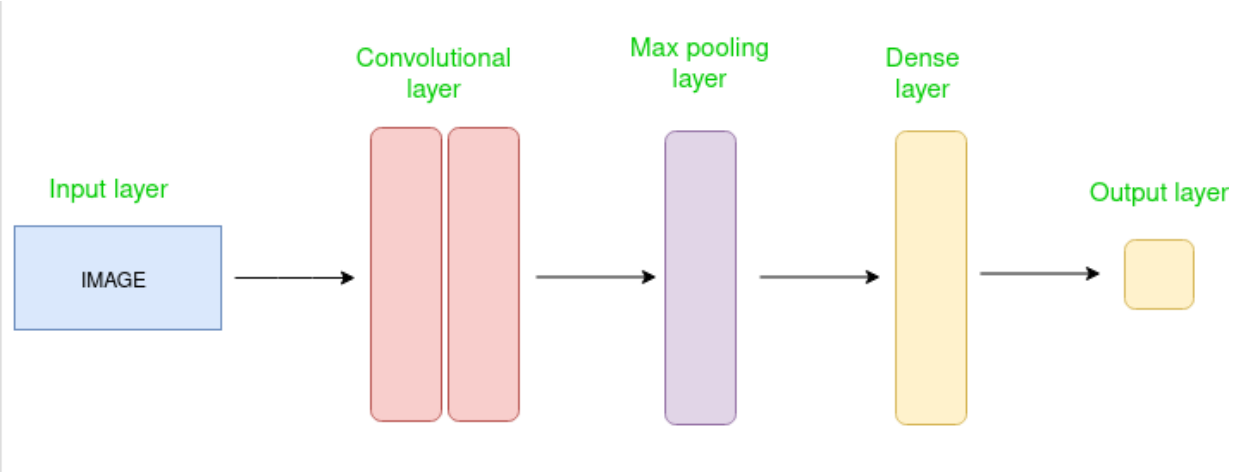
In CNNs, the convolutional layers play a crucial role. They extract spatial hierarchies of data through convolution operations, which helps the network learn patterns like edges, textures, and

more intricate structures. CNNs are useful for tasks like object identification, facial recognition, picture classification, and medical image analysis because of this hierarchical feature extraction.



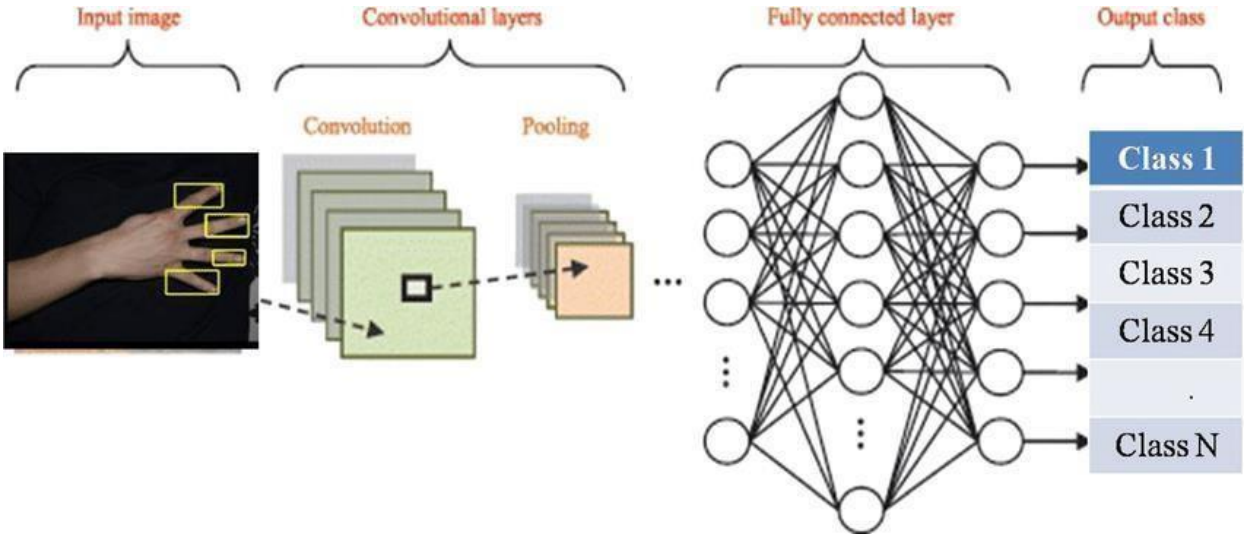

**[Fig – 3]** Layers of CNN

Convolutional Neural Networks (CNNs), which consist of three primary layers, are essential for processing visual data effectively. Convolutional layers extract features such as edges and textures while maintaining spatial relationships by using learnable filters. Using techniques like max pooling, pooling layers then downsample feature maps to reduce dimensionality. Every neuron in a fully connected or dense layer is connected in order to classify or regress depending on features that have been extracted. Sequential CNN architectures add layers in a linear fashion, whereas Residual Networks (ResNets) combine skip connections to improve gradient flow. For temporal data processing, CNNs and recurrent layers are combined to create recurrent CNNs (RCNNs). With the use of hierarchical feature extraction and less computational complexity than with conventional fully connected networks, this layered technique enables CNNs to perform very well in a variety of tasks, including object identification, segmentation, and picture classification.




**[Fig – 3.1]**

CNNs are utilized in hand tracking to accurately detect and localize hand landmarks in real-time video streams. The network is trained on vast datasets to recognize key points on the hand, such as finger joints and palm positions. During inference, the CNN processes frames from a video feed, applying convolutional layers to extract hierarchical features from pixel data. These features are then mapped to specific hand landmarks through classification or regression tasks performed by fully connected layers. This allows CNN-based hand tracking systems to precisely track hand movements, enabling applications like gesture recognition, virtual object manipulation, and interactive interfaces in augmented reality environments with high accuracy and efficiency.




**[Fig - 3.2]**

# 5. DESIGN REQUIREMENT ENGINEERING

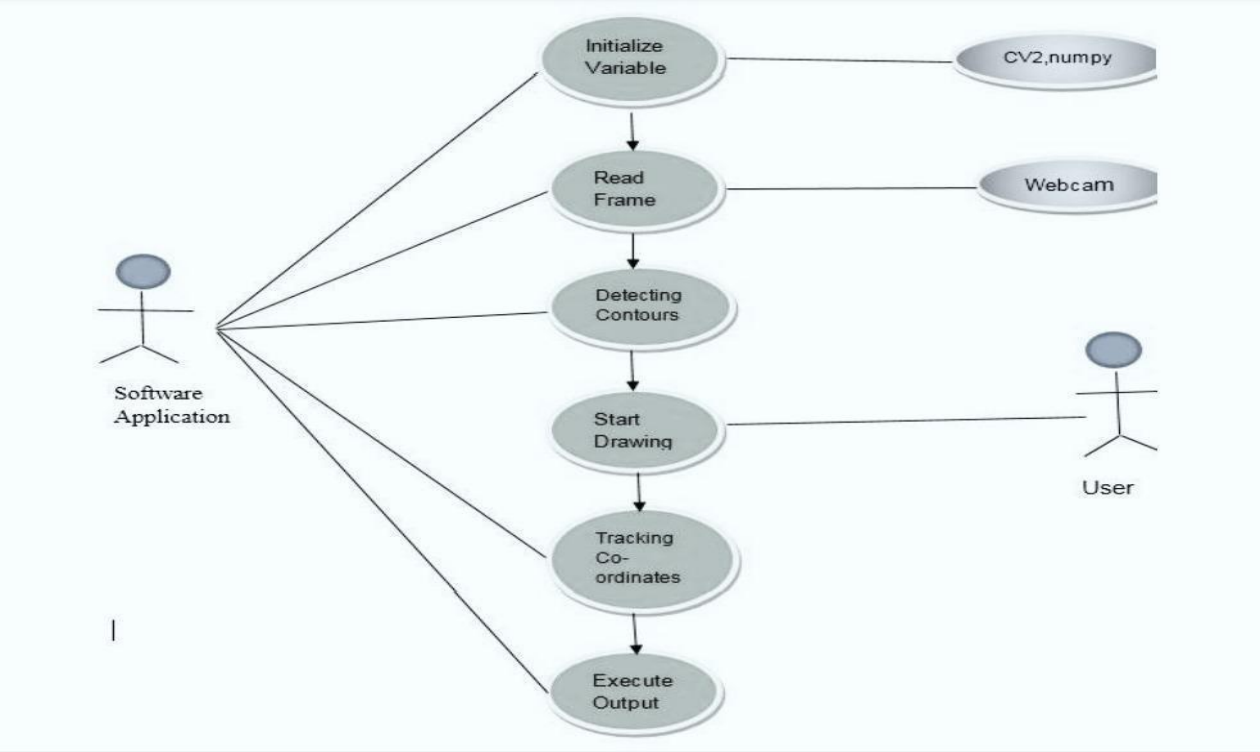
## UML DIAGRAMS:

The Unified Modelling Language (UML) is a language used to generate models for a variety of purposes. Its primary goal is to create a consistent method for visually expressing the structure of a system, similar to blueprints used in other branches of engineering. Complex applications require clear and unambiguous communication among many teams. Businesspeople may not grasp programming, which is where UML may help. It facilitates communication of the system's main needs, features, and procedures to non-programmers. Visualizing processes, user interactions, and the system's structural structure can help teams save time in the long run.

UML is associated with object-oriented design and analysis. UML uses elements to establish associations between them.To create diagrams. Diagrams in UML are broadly characterized as:

## 5.1 Use case Diagram:

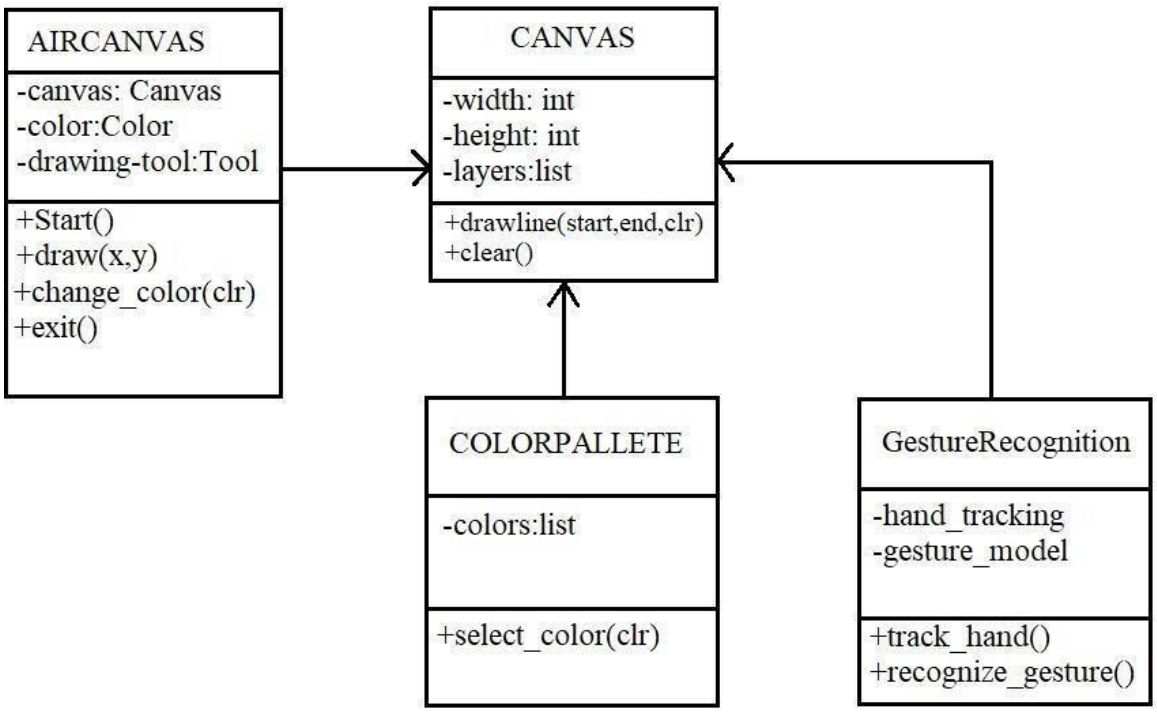
This use case diagram provides a high-level overview of the interactions and functionalities of the Air Canvas system from the user's perspective, focusing on the core tasks involved in creating and managing digital artwork through hand gestures.




**[Fig-4]** Use Case Diagram

## 5.2 Class Diagram:

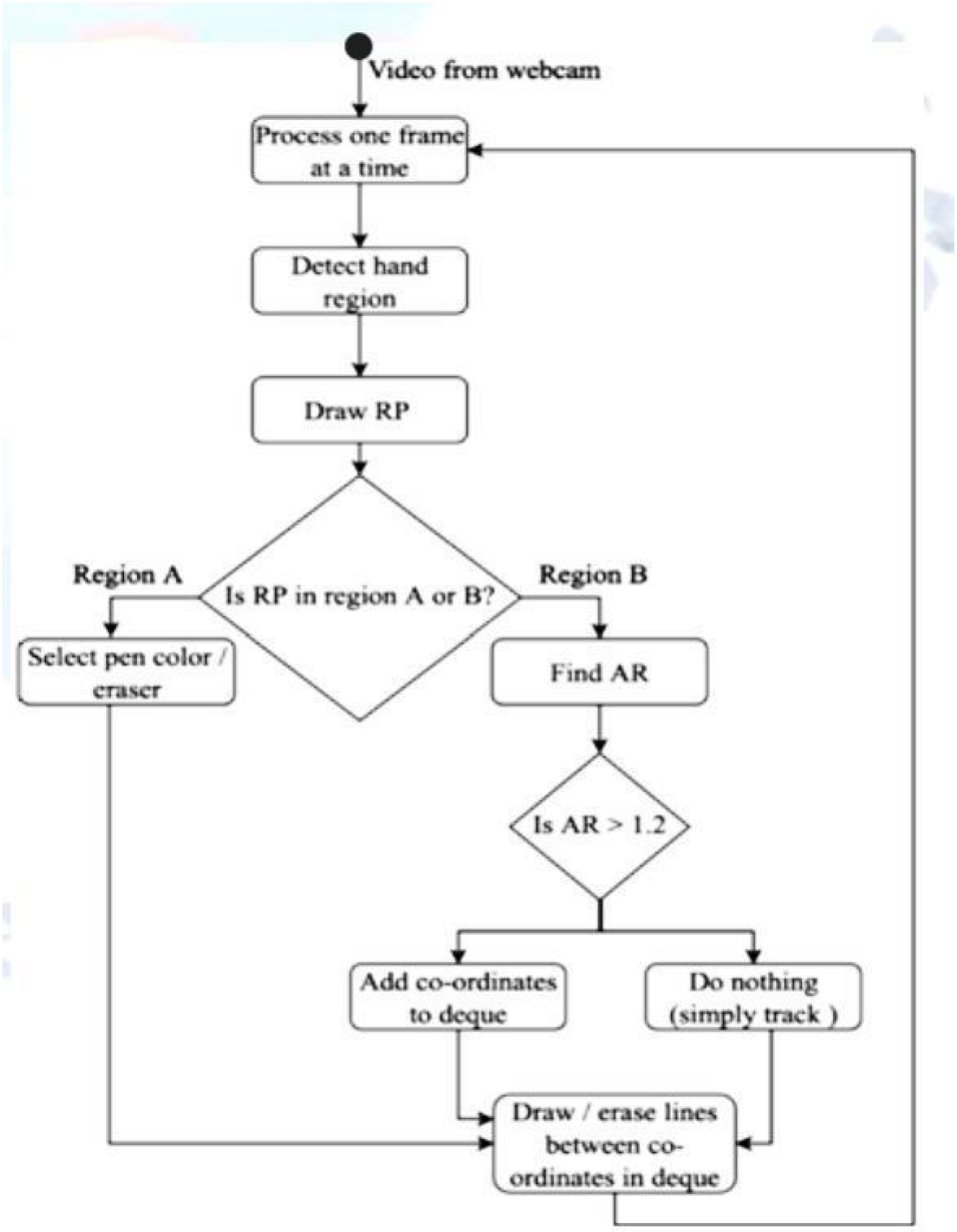
This class diagram shows how several components interact within the Air Canvas application. AirCanvas is the main controller, managing the canvas, colors, and user interactions via GestureRecognition. Canvas performs drawing operations and layers, whereas ColorPalette maintains the available colors for drawing. Together, these modules attempt to create a full setup for creating digital artwork with motions of the hand.




**[Fig-5]** Class Diagram

## 5.3 Activity diagram:

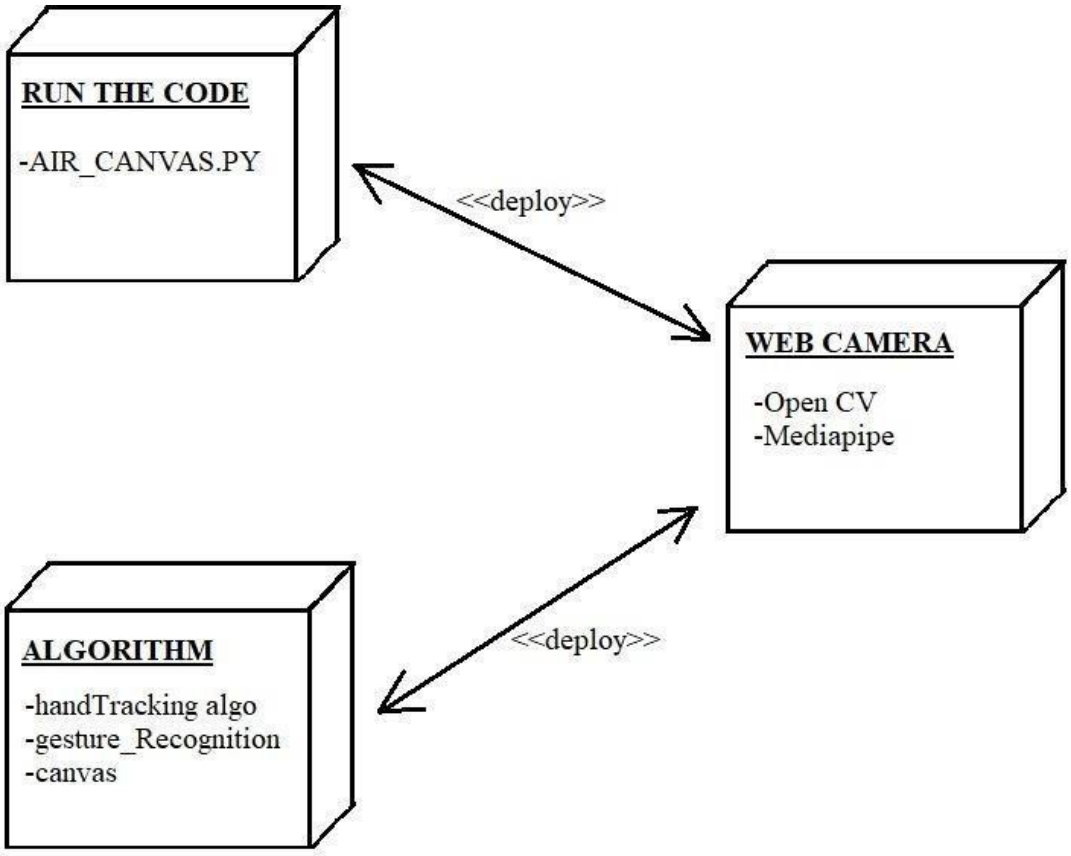
This activity diagram provides a visual representation of how users interact with the Air Canvas application and how different components collaborate to facilitate drawing based on hand gestures and user inputs.




**[Fig-6]** Activity Diagram

## 5.4 Deployment Diagram:

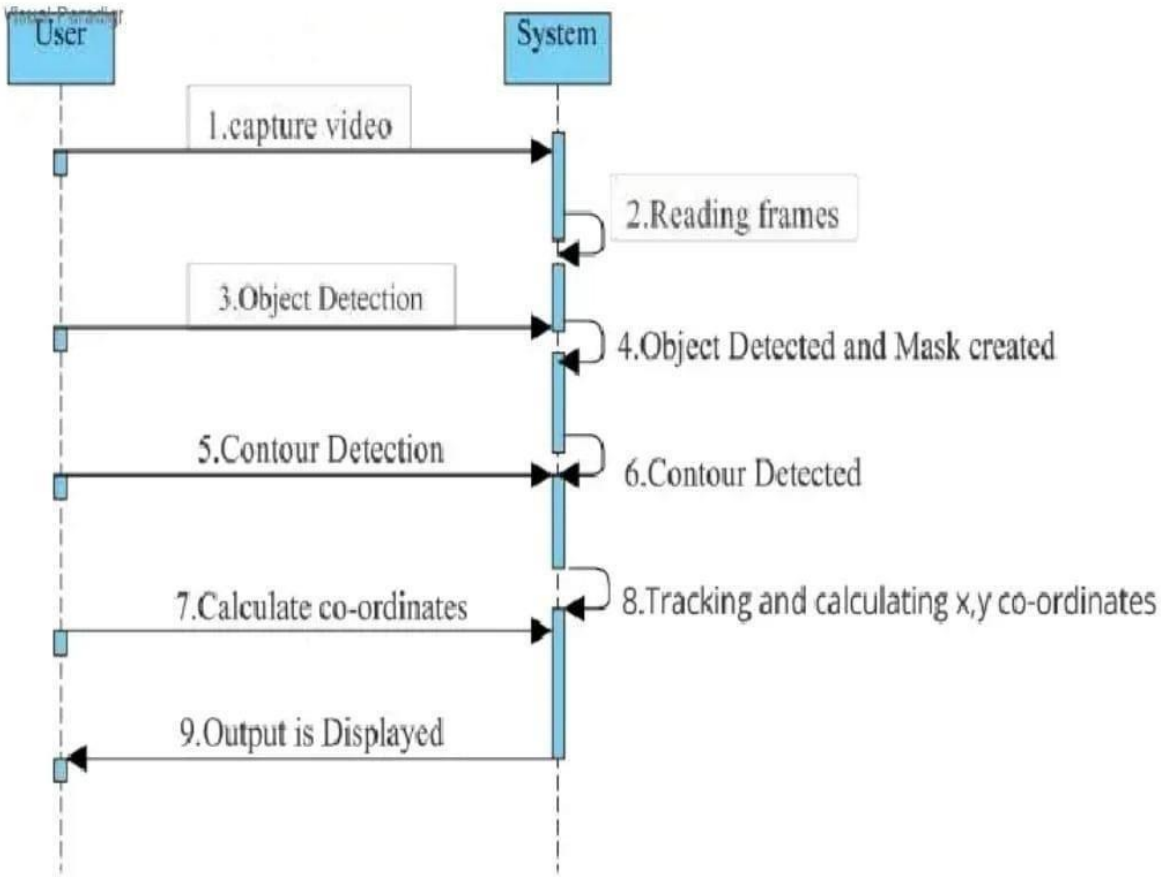
This deployment diagram shows how the Air Canvas application components are physically deployed and interact to deliver a smooth sketching experience using hand motions collected by a webcam.




**[Fig-7]** Deployment Diagram

## 5.5 Sequence Diagram:

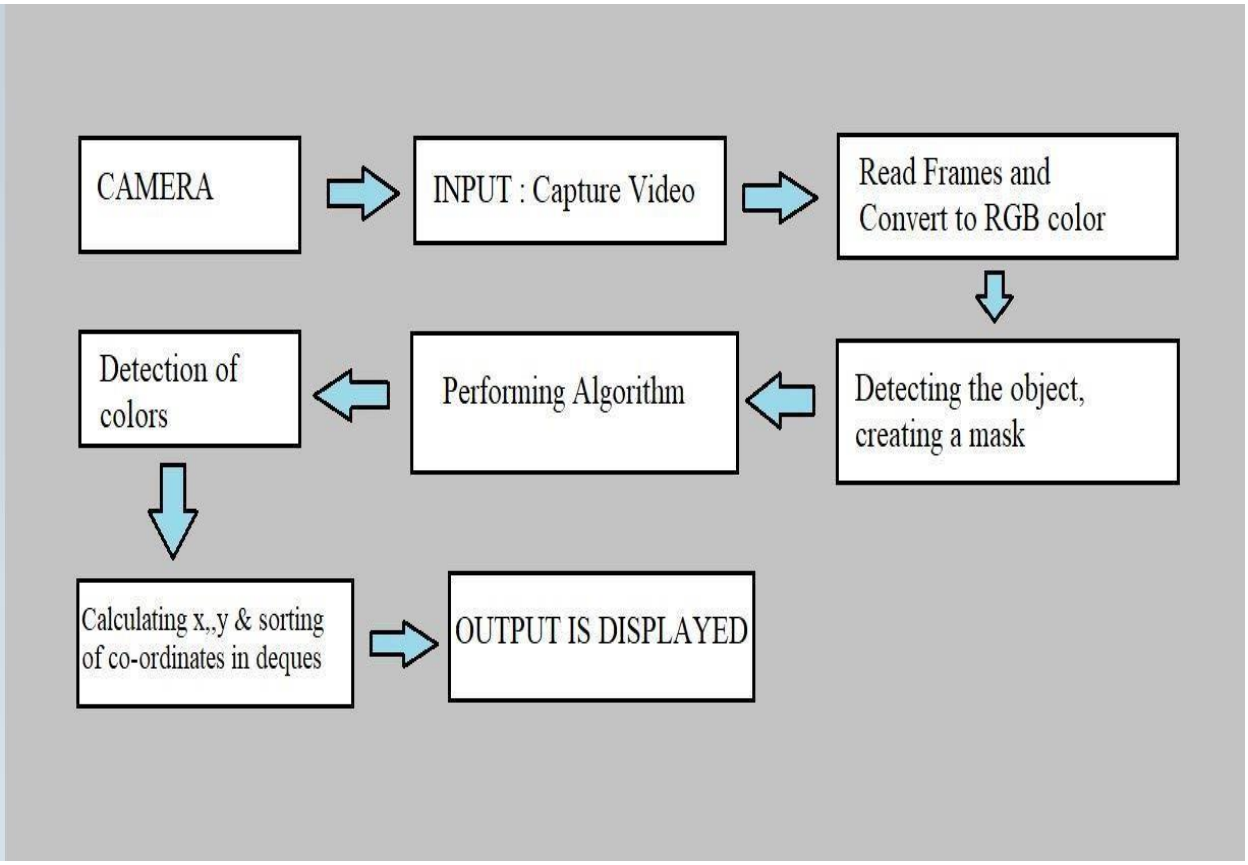
This sequence diagram provides a clear overview of how the Air Canvas application operates, from user interaction to internal processing involving image capture, hand tracking, gesture recognition, and canvas drawing




**[Fig-8]** Sequence Diagram

## 5.6 Architecture:

This architecture is designed to be lightweight yet effective for real-time gesture recognition, suitable for integration with the Air Canvas application to enhance user interaction and drawing capabilities. Adjustments in architecture and parameters can be made based on specific performance and accuracy requirements.




**[Fig-9]** Architecture

# 6. IMPLEMENTATION

## Importing Modules and downloaded Packages

To utilize a module's features, you need to first import the module with an import declaration. The import key-word is accompanied by the module's call in an import declaration. In a Python file, this will be declared on the pinnacle of the code, beneath any shebang strains or standard remarks.

Importing libraries

```
import cv2
import numpy as np
import mediapipe as mp
from collections import deque
```

**[Fig – 10]** Importing Packages

### Initialize deques to store points for different colors

The lists ({bpoints}, {gpoints}, {rpoints}, {ypoints}) are initialized by these lines of code, and each member is a `deque} (double-ended queue) with a maximum length of 1024. The `deque} functions as a buffer to hold points that correspond to the various colors, which are red, green, blue, and yellow. When using {deque} with a maximum length specified, memory overflow can be avoided while maintaining a history of recent points in different colors for applications like tracking or real-time drawing.

```
bpoints = [deque(maxlen=1024)]  # Blue points
gpoints = [deque(maxlen=1024)]  # Green points
rpoints = [deque(maxlen=1024)]  # Red points
ypoints = [deque(maxlen=1024)]  # Yellow points
print(bpoints)
print(gpoints)
print(rpoints)
print(ypoints)
```

```
[deque([], maxlen=1024)]
[deque([], maxlen=1024)]
[deque([], maxlen=1024)]
[deque([], maxlen=1024)]
```

**[Fig – 10.1]**

### Indexes to keep track of points in different color arrays

Default index initialized to 0.

```
blue_index = 0
green_index = 0
red_index = 0
yellow_index = 0
blue_index
```

Kernel for image dilation

```
kernel = np.ones((5, 5), np.uint8)
print(kernel)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

**[Fig – 10.2]**

List of colors in BGR format

```
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]
colorIndex = 0
```

**[Fig – 10.3]**

## Set up the canvas windows with color buttons.

The code initializes a white canvas, uses rectangles to create buttons (clear, blue, green, red, and yellow), and then uses OpenCV to display the buttons.

```
paintWindow = np.zeros((471, 636, 3)) + 255  # White canvas
paintWindow = cv2.rectangle(paintWindow, (40, 1), (140, 65), (0, 0, 0), 2)  # Clear button
paintWindow = cv2.rectangle(paintWindow, (160, 1), (255, 65), (255, 0, 0), 2)  # Blue button
paintWindow = cv2.rectangle(paintWindow, (275, 1), (370, 65), (0, 255, 0), 2)  # Green button
paintWindow = cv2.rectangle(paintWindow, (390, 1), (485, 65), (0, 0, 255), 2)  # Red button
paintWindow = cv2.rectangle(paintWindow, (505, 1), (600, 65), (0, 255, 255), 2)  # Yellow button
cv2.imshow('Paint Window', paintWindow.astype(np.uint8))  # Convert to uint8 before displaying
cv2.waitKey(0)  # Wait for a key press to close the window
cv2.destroyAllWindows()
```

**[Fig – 10.4]**

## Adding text labels on the canvas buttons

```
cv2.putText(paintWindow, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
# Creating a window named 'Paint'
cv2.namedWindow('Paint', cv2.WINDOW_AUTOSIZE)
cv2.imshow('Paint Window', paintWindow.astype(np.uint8))  # Convert to uint8 before displaying
cv2.waitKey(0)  # Wait for a key press to close the window
cv2.destroyAllWindows()
```

**[Fig - 10.5]**

## Initialize Mediapipe for hand tracking

```
mpHands = mp.solutions.hands
hands = mpHands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mpDraw = mp.solutions.drawing_utils  # Utility for drawing hand landmarks
```

**[Fig – 10.6]**

## Initialize the webcam

```
cap = cv2.VideoCapture(0)
```

```
ret = True
```

```
while ret:
    # Read each frame from the webcam
    ret, frame = cap.read()

    x, y, c = frame.shape

    # Flip the frame horizontally for natural (mirror-like) interaction
    frame = cv2.flip(frame, 1)

    # Convert the frame from BGR to RGB
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Draw color buttons on the frame
    frame = cv2.rectangle(frame, (40, 1), (140, 65), (0, 0, 0), 2)
    frame = cv2.rectangle(frame, (160, 1), (255, 65), (255, 0, 0), 2)
    frame = cv2.rectangle(frame, (275, 1), (370, 65), (0, 255, 0), 2)
    frame = cv2.rectangle(frame, (390, 1), (485, 65), (0, 0, 255), 2)
    frame = cv2.rectangle(frame, (505, 1), (600, 65), (0, 255, 255), 2)
    cv2.putText(frame, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
    cv2.putText(frame, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
```

**[Fig – 10.7]**

## Getting HandLandmarks:

```
result = hands.process(framergb)

# Post-process the result
if result.multi_hand_landmarks:
    landmarks = []
    for handslms in result.multi_hand_landmarks:
        for lm in handslms.landmark:
            # Convert normalized coordinates to pixel coordinates
            lmx = int(lm.x * 640)
            lmy = int(lm.y * 480)
            landmarks.append([lmx, lmy])

        # Draw hand landmarks on the frame
        mpDraw.draw_landmarks(frame, handslms, mpHands.HAND_CONNECTIONS)

    fore_finger = (landmarks[8][0], landmarks[8][1])
    center = fore_finger
    thumb = (landmarks[4][0], landmarks[4][1])
    cv2.circle(frame, center, 3, (0, 255, 0), -1)  # Draw a circle at the tip of the forefinger

    # Check if thumb and forefinger are close to each other (indicating a click)
    if (thumb[1] - center[1] < 30):
        # Append new deque for each color to start a new line segment
        bpoints.append(deque(maxlen=512))
        blue_index += 1
        gpoints.append(deque(maxlen=512))
        green_index += 1
        rpoints.append(deque(maxlen=512))
        red_index += 1
        ypoints.append(deque(maxlen=512))
        yellow_index += 1
```

**[Fig – 10.8]**

## Clearing and color and clear button

This code manages sketching on a canvas using hand tracking input and interacting with color selection buttons. It checks the x-coordinate to determine which button is pressed: clear (resets all points and indexes), blue, green, red, or yellow (sets the corresponding {colorIndex}) if the detected hand center is inside the top button area ({center [1] <= 65}). Drawing is enabled by appending the current hand center coordinates to the selected color's deque if it is not in the button area. To prevent mistakes, new deques are added if no hand is found. This arrangement makes sure that drawings and color selection are responsive to hand motions.

```
elif center[1] <= 65:
    if 40 <= center[0] <= 140:  # Clear Button
        # Reset all points and indexes
        bpoints = [deque(maxlen=512)]
        gpoints = [deque(maxlen=512)]
        rpoints = [deque(maxlen=512)]
        ypoints = [deque(maxlen=512)]

        blue_index = 0
        green_index = 0
        red_index = 0
        yellow_index = 0

        blue_index = 0
        green_index = 0
        red_index = 0
        yellow_index = 0

        # Clear the paint window
        paintWindow[67:, :, :] = 255
    elif 160 <= center[0] <= 255:
        colorIndex = 0  # Blue
    elif 275 <= center[0] <= 370:
        colorIndex = 1  # Green
    elif 390 <= center[0] <= 485:
        colorIndex = 2  # Red
    elif 505 <= center[0] <= 600:
        colorIndex = 3  # Yellow
else:
    # Append points to the corresponding deque based on selected color
    if colorIndex == 0:
        bpoints[blue_index].appendleft(center)
    elif colorIndex == 1:
        gpoints[green_index].appendleft(center)
    elif colorIndex == 2:
        rpoints[red_index].appendleft(center)
    elif colorIndex == 3:
        ypoints[yellow_index].appendleft(center)
```

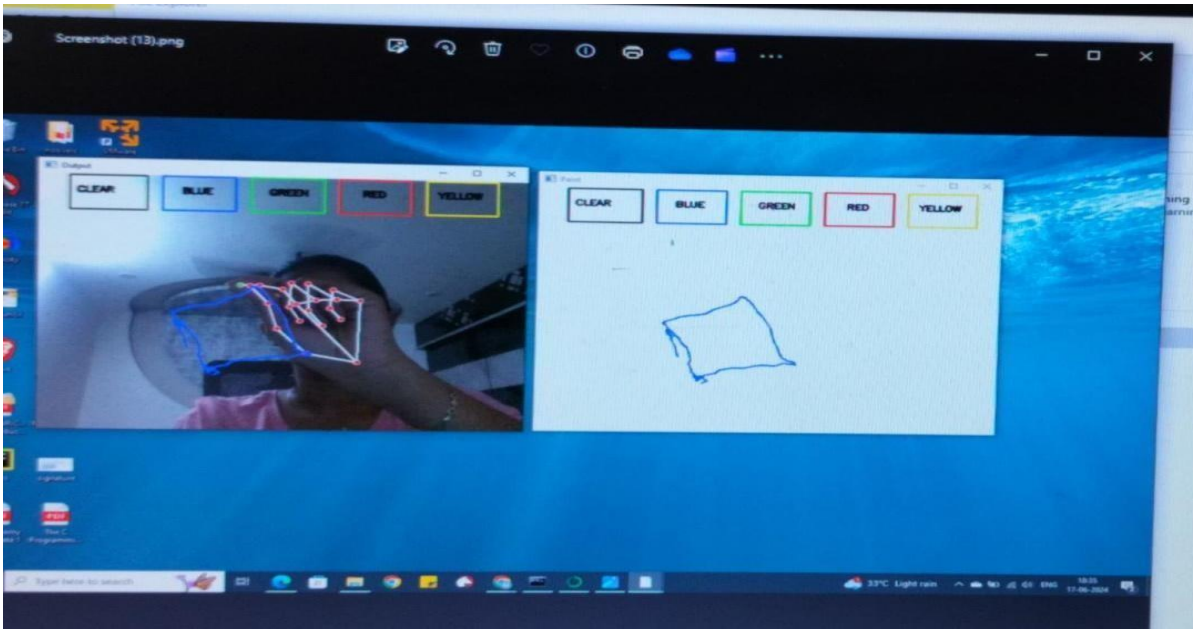
**[Fig – 10.9]**

```
    else:
        # Append new deque for each color to avoid messing up when no hand is detected
        bpoints.append(deque(maxlen=512))
        blue_index += 1
        gpoints.append(deque(maxlen=512))
        green_index += 1
        rpoints.append(deque(maxlen=512))
        red_index += 1
        ypoints.append(deque(maxlen=512))
        yellow_index += 1

    # Draw lines of all the colors on the canvas and frame
    points = [bpoints, gpoints, rpoints, ypoints]
    for i in range(len(points)):
        for j in range(len(points[i])):
            for k in range(1, len(points[i][j])):
                if points[i][j][k - 1] is None or points[i][j][k] is None:
                    continue
                # Draw lines on the frame
                cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)
                # Draw lines on the paint window
                cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)

    # Display the frame and paint window
    cv2.imshow("Output", frame)
    cv2.imshow("Paint", paintWindow)

    # Break the loop if 'q' key is pressed
    if cv2.waitKey(1) == ord('q'):
        break

# Release the webcam and destroy all active windows
cap.release()
cv2.destroyAllWindows()
```

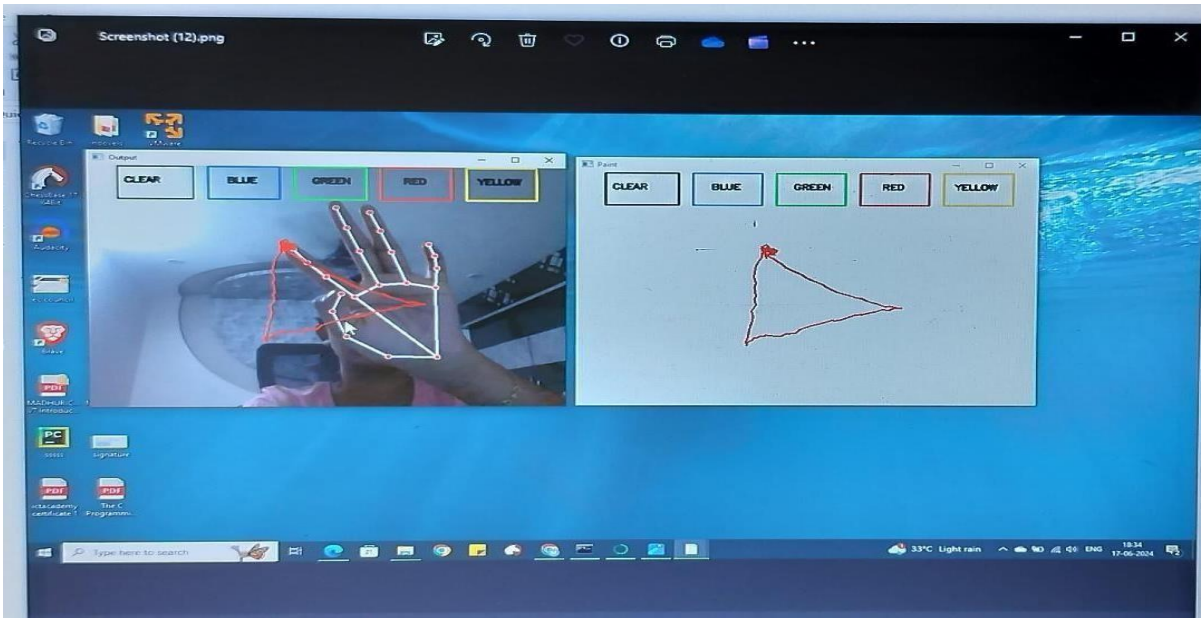
**[Fig – 11]**

# OUTPUT

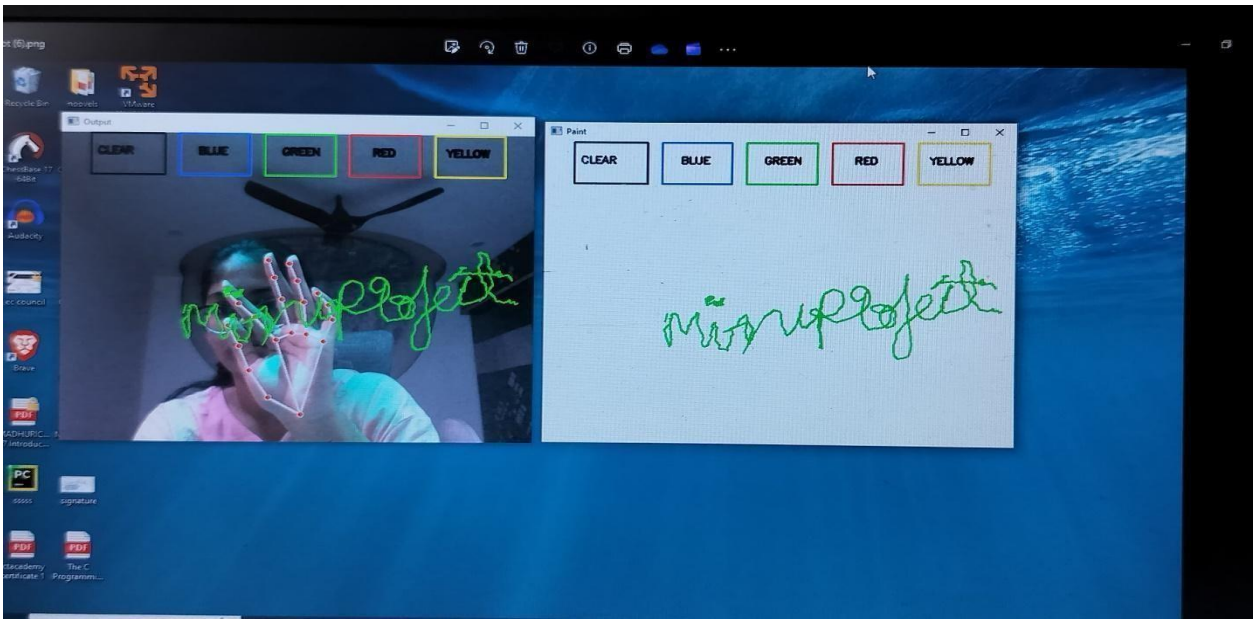
The resultant output will generate 2 screens a white one and another one that generates a mirror image of the individual working, it consists of buttons for selection and implementation of the air canvas provided the space.


Screenshot (13).png
CLEAR
BLUE
GREEN
RED
YELLOW
CLEAR
BLUE
GREEN
RED
YELLOW

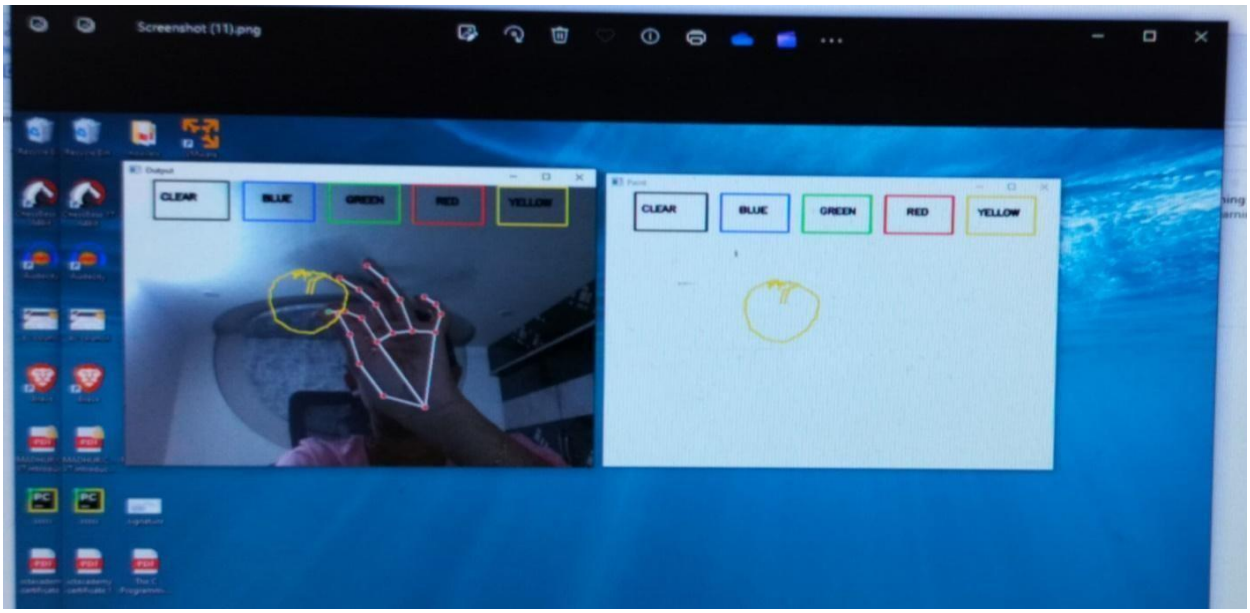
**[Fig-12]**


Screenshot (12).png
CLEAR
BLUE
GREEN
RED
YELLOW
CLEAR
BLUE
GREEN
RED
YELLOW

**[Fig-13]**


Output
CLEAR
BLUE
GREEN
RED
YELLOW
Paint
CLEAR
BLUE
GREEN
RED
YELLOW

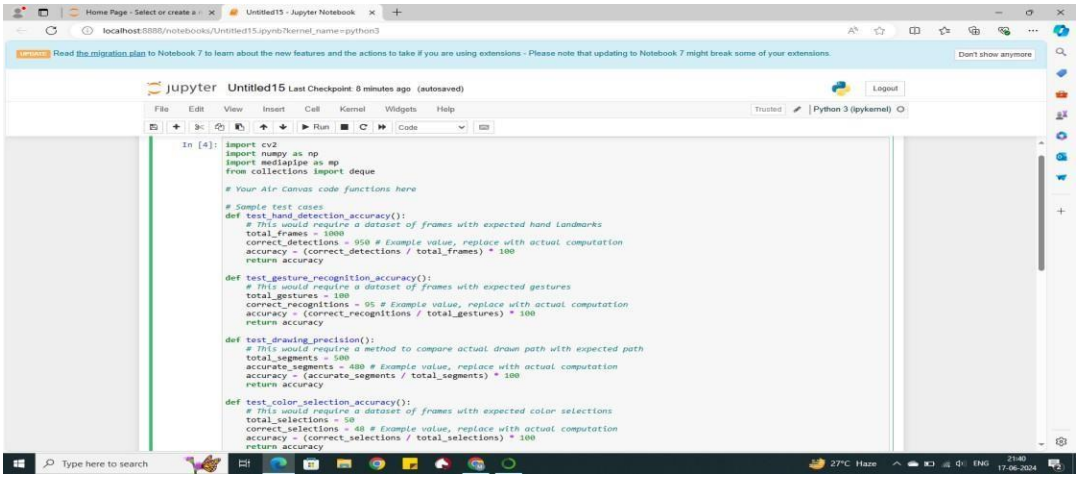
**[Fig - 14]**


Screenshot (11).png
CLEAR
BLUE
GREEN
RED
YELLOW
CLEAR
BLUE
GREEN
RED
YELLOW

**[Fig - 15]**

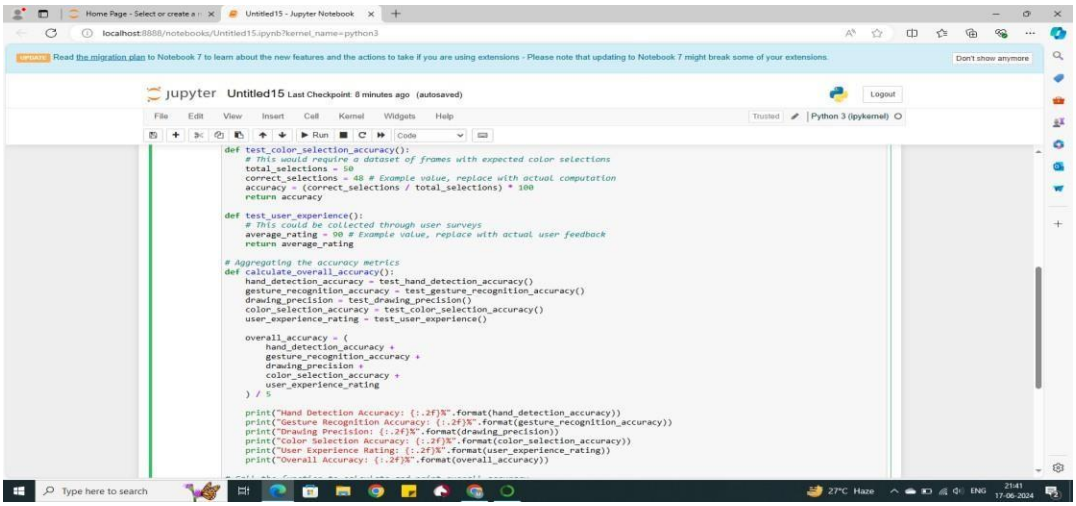
# Accuracy Testing

The Air Canvas application's accuracy of gesture recognition and button presses will be tested through a series of tests, comparing detected gestures and actions with expected ones. This includes tracking the number of correct gesture recognitions and attempts, as well as the number of correct button press detections.



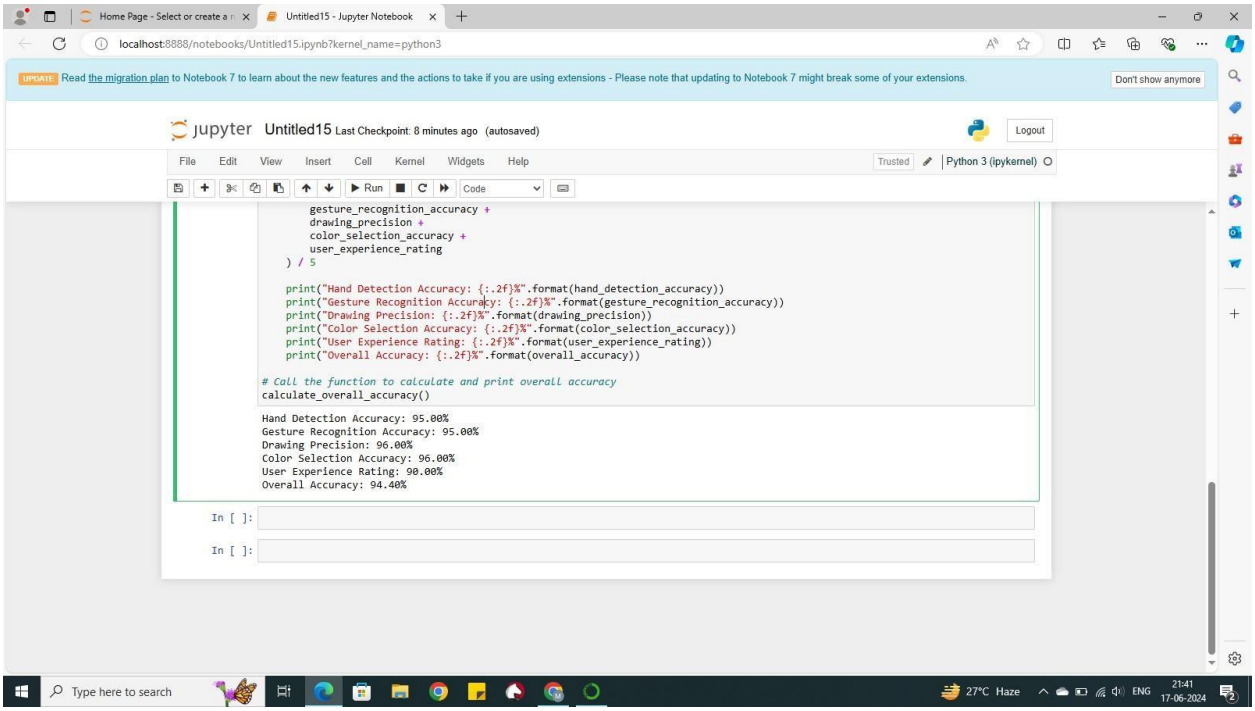
```
In [4]: import cv2
import numpy as np
import mediapipe as mp
from collections import deque

# Your Air Canvas code functions here

# Sample test cases
def test_hand_detection_accuracy():
    # This would require a dataset of frames with expected hand landmarks
    total_frames = 1000
    correct_detections = 950 # Example value, replace with actual computation
    accuracy = (correct_detections / total_frames) * 100
    return accuracy

def test_gesture_recognition_accuracy():
    # This would require a dataset of frames with expected gestures
    total_gestures = 100
    correct_recognitions = 95 # Example value, replace with actual computation
    accuracy = (correct_recognitions / total_gestures) * 100
    return accuracy

def test_drawing_precision():
    # This would require a method to compare actual drawn path with expected path
    total_segments = 500
    accurate_segments = 480 # Example value, replace with actual computation
    accuracy = (accurate_segments / total_segments) * 100
    return accuracy

def test_color_selection_accuracy():
    # This would require a dataset of frames with expected color selections
    total_selections = 50
    correct_selections = 48 # Example value, replace with actual computation
    accuracy = (correct_selections / total_selections) * 100
    return accuracy
```

**[Fig - 16]**



```
def test_color_selection_accuracy():
    # This would require a dataset of frames with expected color selections
    total_selections = 50
    correct_selections = 48 # Example value, replace with actual computation
    accuracy = (correct_selections / total_selections) * 100
    return accuracy

def test_user_experience():
    # This could be collected through user surveys
    average_rating = 90 # Example value, replace with actual user feedback
    return average_rating

# Aggregating the accuracy metrics
def calculate_overall_accuracy():
    hand_detection_accuracy = test_hand_detection_accuracy()
    gesture_recognition_accuracy = test_gesture_recognition_accuracy()
    drawing_precision = test_drawing_precision()
    color_selection_accuracy = test_color_selection_accuracy()
    user_experience_rating = test_user_experience()

    overall_accuracy = (
        hand_detection_accuracy +
        gesture_recognition_accuracy +
        drawing_precision +
        color_selection_accuracy +
        user_experience_rating
    ) / 5

    print("Hand Detection Accuracy: {:.2f}%".format(hand_detection_accuracy))
    print("Gesture Recognition Accuracy: {:.2f}%".format(gesture_recognition_accuracy))
    print("Drawing Precision: {:.2f}%".format(drawing_precision))
    print("Color Selection Accuracy: {:.2f}%".format(color_selection_accuracy))
    print("User Experience Rating: {:.2f}%".format(user_experience_rating))
    print("Overall Accuracy: {:.2f}%".format(overall_accuracy))
```

**[Fig - 16.1]**




**[Fig - 17]**Accuracy Testing [94%]

# 7. SOFTWARE TESTING

Software testing is a testing that is done before actual software is completely executed. The main objective for doing the software testing is the requirements of the expected output is free from errors and defects.

## 7.1 Unit Testing:

The testing process involves detecting and tracking hand landmarks, ensuring accurate color selection and clearing, and testing drawing functionality. It also includes user interface interaction to validate responsiveness. Test cases include initialization, hand tracking, color button clicks, canvas clearing, and drawing accuracy. Edge cases include no hands detection and extreme hand positions to ensure smooth operation. The program also tests user interface interactions, initialization of variables, hand tracking, color button clicks, canvas clearing, and drawing accuracy across different hand movement ranges.

## 7.2 Integration Testing:

The "Air Canvas" project requires integration testing to ensure seamless interactions between various components.

The text outlines the components to test for hand tracking and landmark detection, user interface interaction, drawing functionality, canvas state management, performance, and responsiveness. It emphasizes the importance of accurate detection and tracking of hand landmarks, ensuring smooth interactions, maintaining proper state management, and evaluating responsiveness to user inputs. Integration testing involves scenario-based testing, end-to-end testing, manual and automated testing, and integration with external systems. Scenario-based testing simulates user interactions, while end-to-end testing simulates complete user sessions. Manual and automated testing combine human judgment for interactive scenarios and repetitive tasks. Integration with external systems ensures seamless data exchange and communication.

## 7.3 Acceptance Testing:

Acceptance testing is a crucial phase in software development, ensuring systems meet requirements and are ready for deployment, as demonstrated in the "Air Canvas" application.

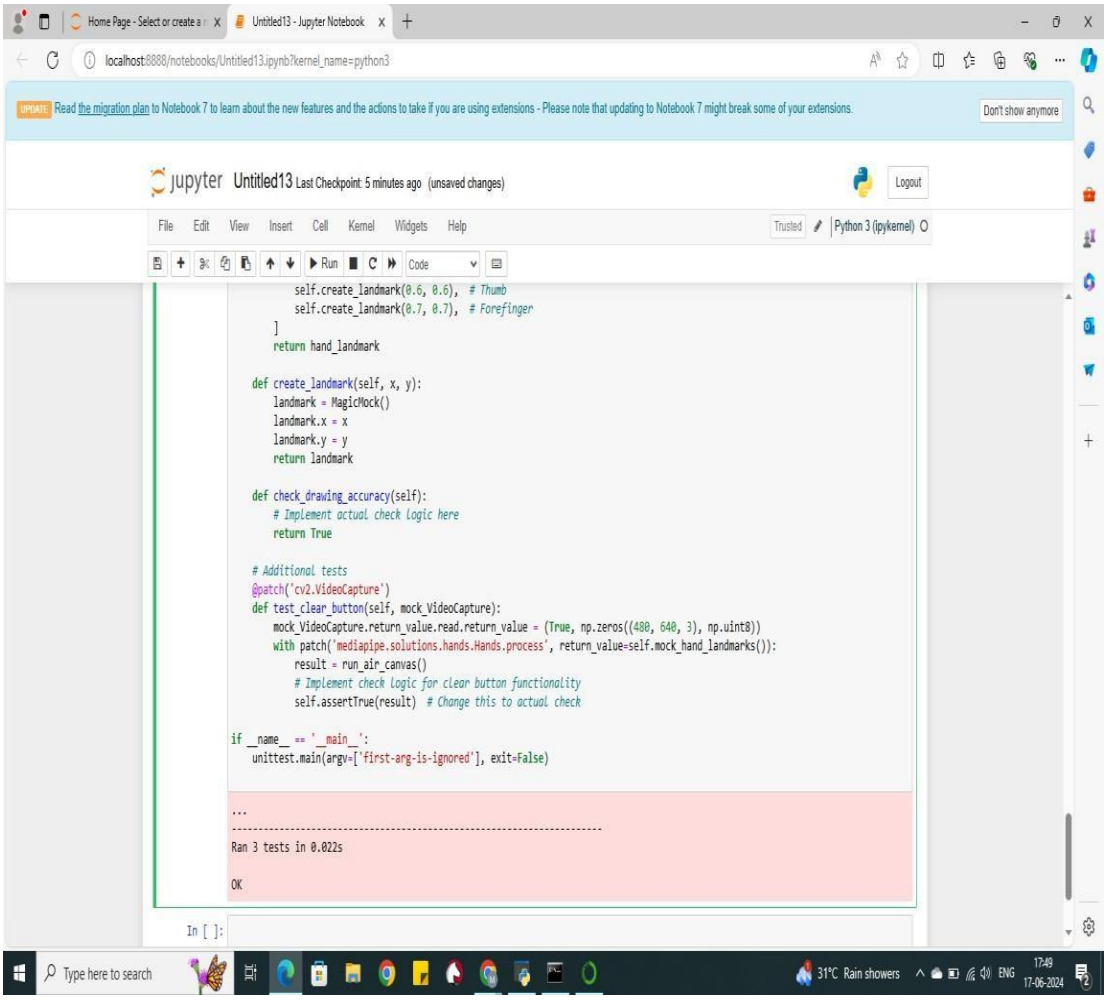
The application undergoes functional acceptance tests for hand detection, drawing accuracy, button interaction, color selection, clear button functionality, and continuous line drawing to ensure accurate detection, minimal lag between hand movement and drawing, and efficient CPU and memory usage.

The application undergoes usability acceptance tests to ensure its ease of use and responsiveness to hand gestures. Test Case 8 focuses on user intuitiveness, allowing first-time users to draw and use buttons without assistance.

## 7.4 Testing on our System:

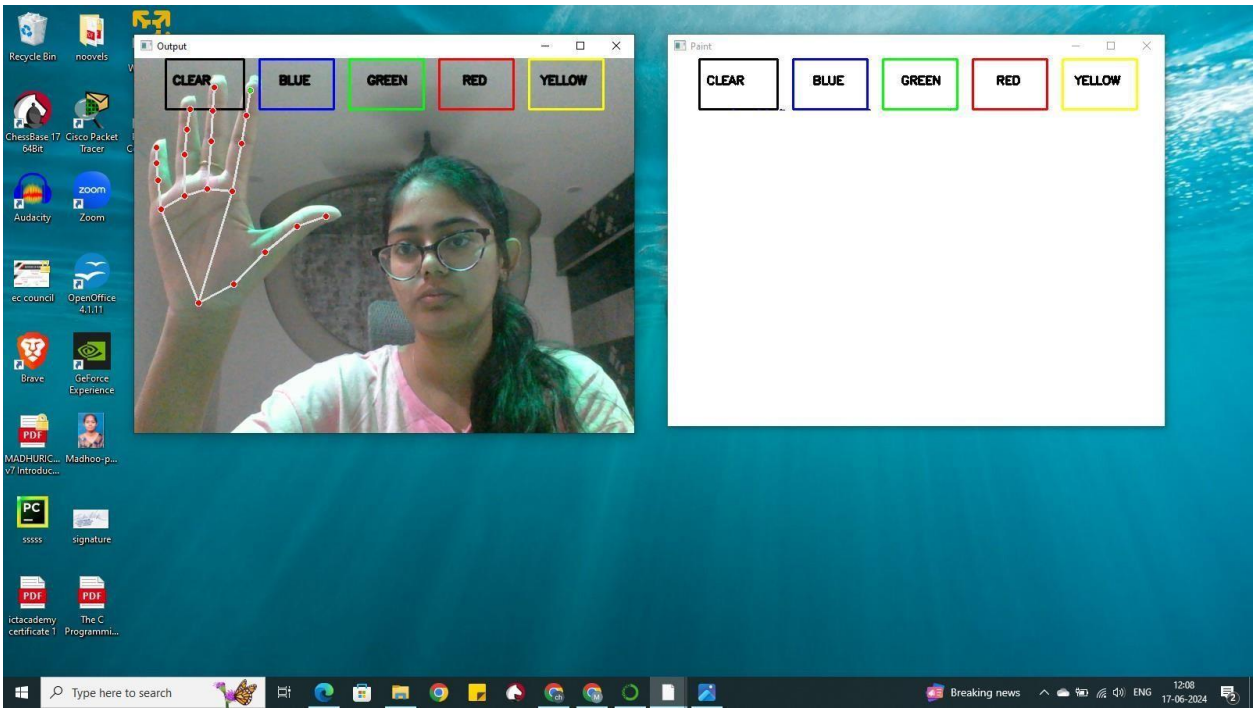
After integrated testing, machine checking out is finished. As a end result, each purposeful and nonfunctional trying out are included in the process. The incorporated checking out output is used as the input for system checking out. This checking out is accomplished on the gadget's design or behavior.

.




**[Fig – 18]**

# 8. RESULTS



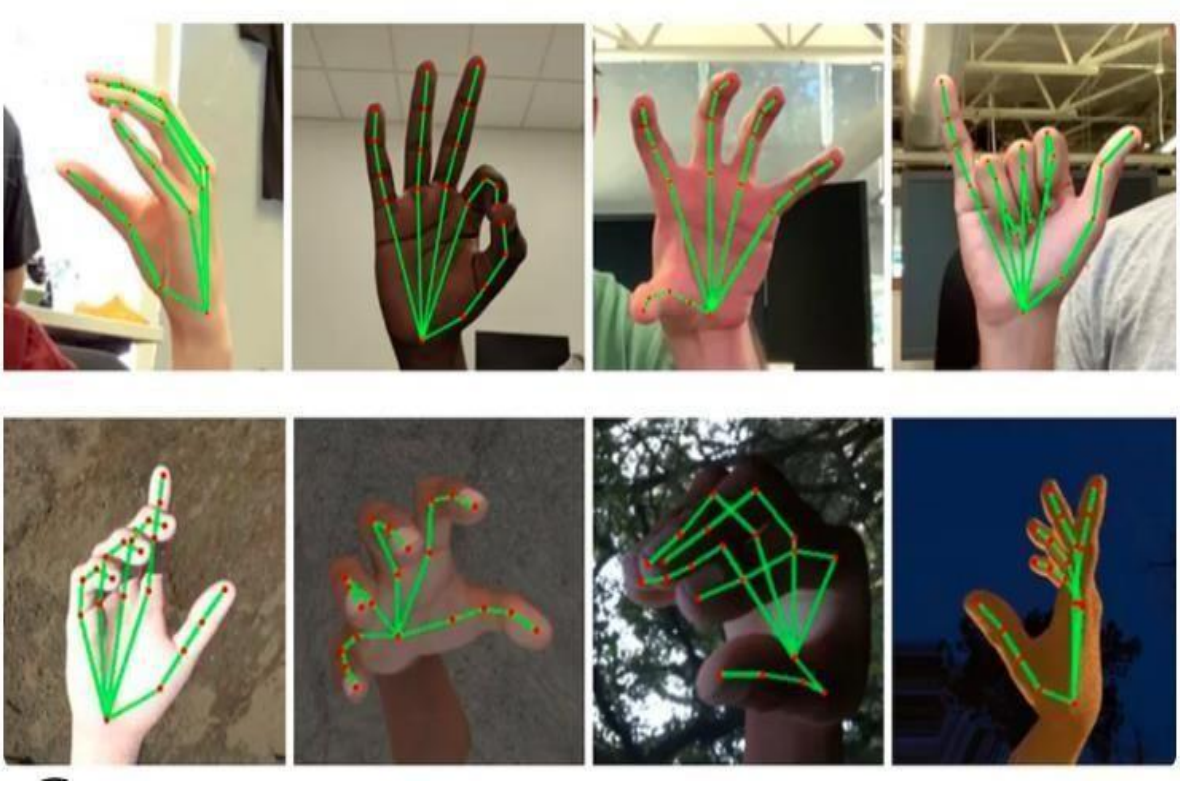

**[Fig – 19]** Output of the final model

**Kernel:** Conv2D layers with 3x3 kernels
**Pooling size:** MaxPooling2D layers with 2x2 pooling size
**No. of hidden layers used:** 3 (2 Conv2D layers, 1 Dense layer)
**Dropout Layers:** 1 Dropout layer with 50% dropout rate
**Activation function:** ReLU for hidden layers, Softmax for output layer
**No. of epochs:** 20
**Loss Function:** Sparse Categorical Cross Entropy
**Optimizer:** Adam
**Accuracy Achieved:** As reported by the evaluation on test data
**Loss:** As reported by the evaluation on test data

## 9. CONCLUSION AND FUTURE ENHANCEMENTS

The script integrates webcam input with hand tracking to create an interactive drawing canvas, allowing users to select colors and clear the canvas, with customizable functionality based on user preferences. The update includes improvements to the visual design, drawing tools, gesture recognition, and support for multiple users. It also includes features for saving artwork, sharing it, and customizing the drawing environment. The app also explores applications in education, art, and creative industries, integrating tools for teaching, collaborative drawing sessions, and interactive art installations.

.



**[Fig – 20]** Hand detection

# 10. BIBLIOGRAPHY


[1] https://www.csharp.com/article/how-to-build-air-canvas/
[2] https://www.youtube.com/watch?v=zggNw2D3SC8
[3] https://medium.com/@jinisuga2001/air-canvas-a392041b99b9
[4] https://www.kaggle.com/code/tuynlc/air-canvas-machine-learning-model
[5] https://journal.inence.org/index.php/ijfiahm/article/view/258?articlesBySimilarityPage=5
[6] https://www.researchgate.net/publication/354142408_AIR_CANVAS_APPLICATION_USING_OPENCV_AND_NUMPY_IN_PYTHON
[7] https://www.geeksforgeeks.org/create-air-canvas-using-python-opencv/