AI BASED HAND GESTURE CONTROL FOR YOUTUBE

A SEMINAR REPORT SUBMITTED

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE OF

BACHELOR OF ENGINEERING
In
Name of the Branch

Computer Science and Engineering

SUBMITTED BY

Sudhanshu Rasotra	Akshay Pawar	Manuj Khajuria
2021A1R001	2021A1R006	2021A1R021

SUBMITTED TO DR. SURBHI GUPTA

Department of Computer Science & Engineering

(Accredited by NBA)

Model Institute of Engineering and Technology (Autonomous)

Jammu, India

Year **2024**

CANDIDATE'S DECLARATION

We , **Sudhanshu Rasotra(2021A1R001), Akshay Pawar (2021A1R006) and Manuj Khajuria (2021A1R021)** hereby declare that the work which is being presented in the seminar report entitled, "**AI Based Hand Gesture Control for Youtube**" in the partial fulfillment of requirement for the award of degree of B.E. (CSE) and submitted in the Department Name, Model Institute of Engineering and Technology (Autonomous), Jammu is an authentic record of our own work carried by us under the supervision of **Dr. Surbhi Gupta** (Designation, MIET) The matter presented in this seminar report has not been submitted in this or any other University / Institute for the award of B.E. Degree.

Sudhanshu Rasotra	2021A1R001
Akshay Pawar	2021A1R006
Manuj Khajuria	2021A1R021

Computer Science and Engineering

Model Institute of Engineering and Technology (Autonomous)

Kot Bhalwal, Jammu, India

(NAAC "A" Grade Accredited)

Date: 31/05/2024

CERTIFICATE

Certified that this seminar report entitled **"AI Based Hand Gesture Control for YouTube"** is the bonafide work of "**Sudhanshu Rasotra(2021A1R001), Akshay Pawar (2021A1R006) and Manuj Khajuria (2021A1R021) of 6th Semester, Computer Science and Engineering, Model Institute of Engineering and Technology (Autonomous), Jammu",** who carried out the seminar work under my supervision during May, 2024.

Dr. Surbhi Gupta
Coordinator
Assistant Professor
CSE, MIET

ACKNOWLEDGEMENTS

As a part of this semester's curriculum at MIET we got the opportunity to work on the mini project "AI Based Hand Gesture Control for YouTube" which aided me an unsurmountable amount in enhancing my practical experience with the working of Machine Learning and various steps involved.

First and foremost, we would like to pay our gratitude to Dr. Surbhi Gupta, project coordinator, for guiding us through this phase and for being an excellent mentor throughout the course of this project evaluation.

It is also an honor with which we express my overwhelming gratitude towards Prof. Ankur Gupta (Director, MIET) and Prof. Navin Upadhayay (Dean Academics).

As we've always been and always be, we remain deeply thankful to each other group member without whom this result could never have been achieved. Lastly, we will have to express our immense gratitude to or parents as well as our teachers who guided us and helped us get through this project whenever we hit a dead end. The reason we are at where we are today, is all but them. Never a single moment did they not spare when it came to everything that concerned us and our life, they selflessly acted as the pillars of our entire journey up until now.

Sudhanshu Rasotra	**Akshay Pawar**	**Manuj Khajuria**
2021A1R001	**2021A1R006**	**2021A1R021**

ABSTRACT

This project focuses on developing a hand gesture recognition system for controlling YouTube videos using a Convolutional Neural Network (CNN) and computer vision techniques. The system leverages a webcam to capture live video feeds, detects hand gestures in real-time, and translates these gestures into specific commands for YouTube video control.

The system introduces a control delay mechanism to prevent rapid and unintended command execution, ensuring a smooth and user-friendly interaction. The overall architecture combines real-time computer vision, deep learning, and automation to provide an intuitive and hands-free method of interacting with YouTube.

This project demonstrates the potential of integrating computer vision and deep learning for gesture-based control systems, offering an innovative solution that enhances user experience and accessibility in multimedia applications. The modular design allows for easy adaptation to other gesture-based control scenarios beyond YouTube, showcasing the versatility and scalability of the approach.

Contents

LIST OF FIGURES

ABBREVIATIONS USED

CNN: Convolutional Neural Networks

GUI: Graphical User Interface

CV: Computer Vision

OS: Operating System

MP: Mediapipe

H5: HDF5 (Hierarchical Data Format version 5)

Pyautogui: Python Automation GUI

FPS: Frames Per Second

ML: Machine Learning

TF: TensorFlow

CHAPTER 1

PYTHON PROGRAMMING

Introduction 1.1

Python is a high-level, versatile, and dynamically typed programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, Python has gained immense popularity among developers, data scientists, and engineers due to its ease of use and extensive standard libraries

Python is a multiparadigm programming language, supporting procedural, object-oriented, and functional programming styles. It can be used for various applications, including web development, data analysis, artificial intelligence, scientific computing, automation, and more. Python provides a vast collection of libraries and frameworks that simplify complex tasks. For example, NumPy and Pandas are used for data manipulation and analysis, Flask and Django for web development, TensorFlow and PyTorch for machine learning, and Matplotlib for data visualization.

Python is an interpreted language, which means that code written in Python is executed line by line by the Python interpreter. This feature facilitates rapid development and debugging. It is platform-independent, allowing developers to write code on one platform and run it on various operating systems without modification.

Python has become the go-to language for data scientists and AI researchers. Libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and PyTorch provide powerful tools for data analysis, machine learning, deep learning, and natural language processing. Its simplicity and ease of integration have made it the preferred choice for professionals working with complex datasets and developing cutting-edge AI applications.

Syntax 1.2

All Python programs do not follow the very same syntax with a defined beginning and end as a result of it being a high level language. There are, however, some important syntaxes for various python operations that are the most used.

1) Print statement

print("Hello, World!")

2) Assigning values to variables

variable_name = value

These values may be:

- Integer: 42
- Float: 3.14
- String: "Hello"
- Boolean: True or False

3) Arithmetic operations:

- addition = a + b
- subtraction = a - b
- multiplication = a * b
- division = a / b

4) Conditional statements:

- if condition:
- else:

5) Loops:

- for variable in iterable:
- while condition:

6) Data collection types:

- my_list = [1, 2, 3, 4, 5]
- my_dict = {"key": "value", "name": "Alice"}
- my_tuple = (1, 2, 3, 4, 5)
- my_set = {1, 2, 3, 4, 5}

7) Functions:

```
def function_name(parameters):
```

8) Classes and Objects:

```
class ClassName:
    def __init__(self, parameters):
    def class_method(self, parameters):
obj = ClassName(arguments)
```

9) Comments:

```
# This is a single-line comment
"""
This is a multi-line
comment
"""
```

Features 1.3

Python is a language rich in features that define its significance as a high level language and are the reasons behind its user friendly nature.

Easy to Read and Learn:

Python's simple and clean syntax emphasizes readability and reduces the cost of program maintenance. This feature makes it an excellent choice for beginners.

Interpreted Language:

Python is an interpreted language, which means that code written in Python is executed line by line by the Python interpreter. This leads to faster development as there is no need for compilation.

High-Level Language:

Python abstracts low-level details, providing a high-level interface to developers. This simplifies complex algorithms and allows developers to focus on solving problems.

Object-Oriented:

Python supports object-oriented programming (OOP) principles, enabling the creation and use of objects and classes. This paradigm encourages modularity, reusability, and a more organized code structure.

Dynamically Typed:

Python is dynamically typed, meaning that variable types are interpreted during runtime. Developers do not need to specify variable types explicitly, making the code more flexible and readable.

Interoperability:

Python can be integrated with other languages like C, C++, and Java. This feature allows developers to utilize existing code libraries and tools from these languages.

Large Standard Library:

Python comes with a vast collection of modules and libraries that simplify various programming tasks. These libraries cover areas such as file I/O, regular expressions, networking, databases, and more.

Strong Support for Data Science and Machine Learning:

Python is a preferred language in the fields of data science, machine learning, and artificial intelligence, thanks to powerful libraries like NumPy, pandas, scikit-learn, TensorFlow, and PyTorch.

Applications 1.4

Thanks to its high range of versatility, Python is used across various platforms for a wide variety of tasks and operations.

Data Science and Analysis:

Python, along with libraries like Pandas, NumPy, and SciPy, is widely used for data analysis, manipulation, and visualization. Tools like Jupyter Notebooks facilitate interactive data analysis and exploration.

Machine Learning and Artificial Intelligence:

Python is the preferred language for machine learning and AI projects. Libraries like TensorFlow, PyTorch, and scikit-learn provide powerful tools for developing machine learning models and neural networks.

Web Development:

Python frameworks like Django, Flask, and Pyramid enable developers to build robust and scalable web applications. Python's simplicity and readability are advantageous in web development projects.

Scientific Computing:

Python is extensively used in scientific computing, simulations, and mathematical modeling. Libraries like SciPy and NumPy offer support for scientific computations, making it a favorite among researchers and scientists.

Natural Language Processing (NLP):

Python's NLTK (Natural Language Toolkit) and SpaCy libraries are widely used in natural language processing tasks such as text analysis, sentiment analysis, and language translation.

Game Development:

Python, with libraries like Pygame, is used in game development for creating 2D games and prototypes. Its ease of use and rapid development capabilities are advantageous in game development projects.

Internet of Things (IoT):

Python is popular in IoT projects due to its simplicity and support for various hardware platforms. Libraries like MicroPython enable developers to program microcontrollers and IoT devices using Python.

Web Scraping:

Python's libraries such as BeautifulSoup and Scrapy are used for web scraping tasks, allowing developers to extract data from websites and analyze it for various purposes.

Automation and Scripting:

Python is widely used for automation tasks, scripting, and workflow management. It simplifies tasks like file manipulation, data processing, and system automation.

Network Programming:

Python is used for network programming tasks such as socket programming, network protocol development, and building network applications. Libraries like Twisted provide support for asynchronous network programming.

Desktop GUI Applications:

Python offers GUI frameworks like Tkinter, PyQt, and Kivy for developing desktop applications with graphical user interfaces. These frameworks are used to create applications with interactive interfaces.

16

CHAPTER 2

JUPYTER LAB

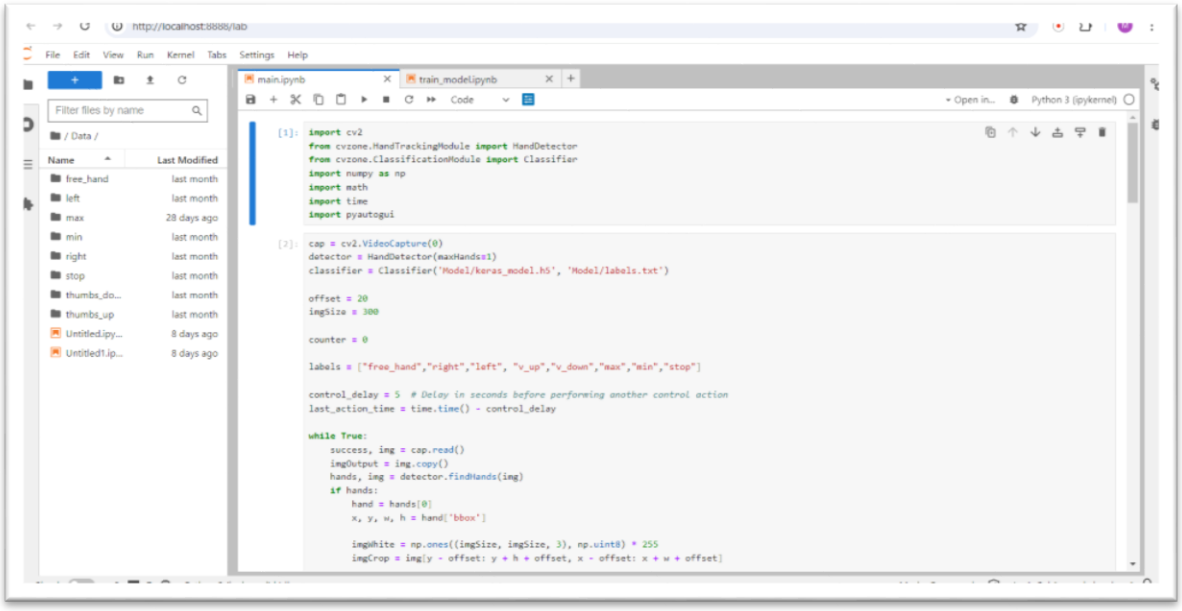
Introduction 2.1

Jupyter Lab is an open-source, interactive web-based development environment for data science, machine learning, and scientific computing. It is part of the larger Jupyter Project, which started with the IPython project in 2001 and has since evolved into a powerful ecosystem for interactive computing and data analysis.

Jupyter Lab provides an intuitive and flexible interface that allows users to work with various document formats, including Jupyter Notebooks, text files, images, interactive plots, and custom components, all within a single web application. It supports multiple programming languages, with Python being the most commonly used one. Other languages such as R and Julia are also supported.

While Python is the most popular language used in Jupyter Lab, it supports various programming languages through kernels. Kernels are language-specific execution environments that allow users to run code in different languages within the same Jupyter notebook.

Jupyter Lab has become an essential tool in the data science and research communities due to its flexibility, interactivity, and collaborative features. It provides a convenient environment for prototyping, experimenting, and sharing data-driven insights, making it a valuable asset for both beginners and experienced professionals in the field of scientific computing.



Jupyter Lab UI

Features 2.2

Jupyter lab comes with an impressive range of user friendly features that make the workflow dynamic as well as versatile. These key features available in Jupyter lab are as follows:

Interactive Widgets: Jupyter Lab supports interactive widgets that enable the creation of dynamic user interfaces for data exploration and analysis. These widgets include sliders, buttons, dropdowns, and other interactive components, enhancing the interactivity of notebooks.

File Management: Jupyter Lab provides a file browser and an integrated text editor, allowing users to manage files, edit code, and organize their projects within the same environment. It also supports Git integration for version control.

Visualization and Plotting: Jupyter Lab supports a wide range of data visualization libraries, including Matplotlib, Plotly, Bokeh, and more. Users can create interactive plots, charts, and graphs directly within notebooks for exploratory data analysis.

Console and Terminal Integration: Jupyter Lab includes interactive Python consoles and terminal interfaces, allowing users to execute code snippets, run scripts, and perform system commands directly from the interface.

Notebook Conversion: Jupyter Lab allows users to convert notebooks into various formats, including HTML, PDF, slideshows, and more. This feature is useful for sharing research findings and presentations with others.

Collaborative Capabilities: Jupyter Lab can be run on remote servers, enabling collaborative work on shared notebooks. Users can collaborate in real-time, making it a valuable tool for team-based data science projects.

Support for Data Science Libraries: Jupyter Lab seamlessly integrates with popular data science libraries and frameworks such as Pandas, NumPy, SciPy, scikit-learn, TensorFlow, and PyTorch, providing a cohesive environment for data analysis and machine learning.

These features collectively make Jupyter Lab a powerful and versatile tool for data scientists, researchers, educators, and developers working on interactive computing, data analysis, and scientific research.

Applications 2.3

Being as feature richas it is, its only natural for Jupyter lab to have a humongous number of applications as weel. Some of them are listed below:

Interactive Notebooks: Jupyter Lab provides a rich interactive computing environment through Jupyter notebooks. Notebooks support code execution, rich text, mathematical expressions, inline plots, interactive widgets, and more, making them ideal for data analysis, visualization, and experimentation.

Multi-Language Support: Jupyter Lab supports multiple programming languages, including Python, R, Julia, and more. Each language is supported by specific kernels, allowing seamless integration and execution of code written in different languages within the same environment.

Flexible and Extensible Interface: Jupyter Lab features a flexible, extensible interface that supports arranging documents, code consoles, and data visualizations in a customizable manner. Users can split panes, drag and drop components, and organize their workspace according to their preferences.

Rich Text Support: Jupyter Lab allows the incorporation of rich text elements in notebooks using Markdown, LaTeX, and HTML, enabling users to create interactive documents with detailed explanations, equations, and multimedia content.

Data Analysis and Visualization: Jupyter Lab is extensively used in data analysis and visualization tasks. Data scientists and analysts utilize Jupyter notebooks to explore datasets, perform statistical analysis, and create interactive visualizations using libraries like Matplotlib, Plotly, and Seaborn.

Machine Learning and Data Modeling: Jupyter Lab provides an interactive environment for developing machine learning models. Data scientists can experiment with algorithms, preprocess data, and visualize model performance, enabling efficient prototyping and development in libraries like scikit-learn, TensorFlow, and PyTorch.

Scientific Research: Researchers in fields such as physics, biology, and engineering use Jupyter Lab for simulations, data analysis, and sharing research findings. Its support for mathematical expressions, interactive widgets, and rich text makes it ideal for scientific communication and collaboration.

Data Cleaning and Transformation: Data preprocessing tasks, including cleaning, transformation, and feature engineering, can be efficiently performed using Jupyter Lab. Analysts can document their steps and visualize data distributions, aiding in data preparation for further analysis.

Exploratory Data Analysis (EDA): Jupyter Lab facilitates exploratory data analysis by allowing analysts to interactively explore datasets, generate summary statistics, create interactive charts, and identify patterns and outliers. EDA is crucial for understanding data characteristics before diving into in-depth analysis.

Presentations and Reports: Jupyter Lab allows users to create dynamic presentations and reports by combining code, visualizations, and explanatory text. The ability to create slideshows directly from notebooks makes it a popular choice for sharing insights and results in a presentation format.

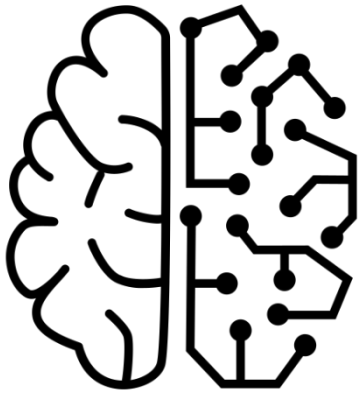
Collaborative Research: Researchers and scientists can collaborate on projects using Jupyter Lab. Multiple researchers can work on the same project simultaneously, enabling collaborative data analysis and research.

CHAPTER 3

MACHINE LEARNING WITH AI

Introduction 3.1

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and models that allow computers to improve their performance on a specific task through experience or data. In other words, machine learning enables computers to learn from data and make predictions or decisions without being explicitly programmed.



There are several types of machine learning algorithms, including:

Supervised Learning: This type of machine learning involves training a model on labeled data, where the algorithm learns to map input data to the corresponding output labels. It's called "supervised" because the process of an algorithm learning from the training dataset can be compared to a teacher supervising the learning process.

Unsupervised Learning: Unsupervised learning deals with unlabeled data. The system tries to learn the patterns and the structure from the data without any supervision. Clustering and association are common types of unsupervised learning algorithms.

Reinforcement Learning: In reinforcement learning, an agent learns how to behave in an environment by performing actions and receiving rewards. Its objective is to learn to act in a way that will maximize its expected long-term rewards. Reinforcement learning is a powerful paradigm for training agents to make decisions in complex environments and has seen remarkable success in various domains. However, it also presents unique challenges that require sophisticated algorithms and significant computational resources to address.

When working with machine learning and AI, several key steps are involved in the development process:

Problem Definition: Clearly define the problem you want to solve with AI. Understand the goals and objectives of the AI system.

Data Collection: Gather and prepare the relevant data that will be used to train and test the machine learning algorithms.

Feature Engineering: Select and transform the features (variables) in the data to create meaningful inputs for the machine learning algorithms.

Model Selection: Choose an appropriate machine learning algorithm or model based on the nature of the problem (e.g., regression, classification, deep learning, etc.).

Training: Train the selected model using the training data to learn the patterns and relationships in the data.

Evaluation: Evaluate the model's performance using validation data or techniques like cross-validation to ensure it generalizes well to unseen data. Integrate the trained model into the AI system, allowing it to make predictions or decisions based on new, unseen data.

Deployment: Integrate the trained model into the AI system, allowing it to make predictions or decisions based on new, unseen data.

Applications 3.2

As a result of powerful analyzation and deductive abilities, Machine learning is no short of applications in today's era. Here are some key applications of machine learning:

1. **Recommendation Systems**:

Movie/TV Show Recommendations: Platforms like Netflix and Hulu use machine learning to recommend content based on user preferences and viewing history.

2. **Natural Language Processing (NLP)**:

Sentiment Analysis: Determining the sentiment behind a piece of text, commonly used in social media monitoring and customer feedback analysis.

Language Translation: Machine translation services like Google Translate use machine learning algorithms to improve accuracy.

Chatbots: AI-driven chatbots use machine learning to understand and respond to human queries.

3. **Image and Video Analysis:**

Image Recognition: Machine learning algorithms can be trained to recognize objects, people, or scenes within images.

4. **Predictive Analytics**:

Sales Forecasting: Predicting future sales based on historical data, helping businesses optimize inventory and resources.

Financial Forecasting: Predicting stock prices, currency exchange rates, and other financial indicators.

Demand Prediction: Predicting demand for products or services to optimize supply chain management.

5. **Healthcare**:

Disease Prediction: Machine learning algorithms can analyze patient data to predict diseases or conditions such as diabetes or heart diseases.

Medical Image Analysis: Analyzing medical images like X-rays and MRIs for early detection of diseases or abnormalities.

Drug Discovery: Accelerating drug discovery by analyzing molecular structures and predicting potential drug candidates.

CHAPTER 4

CNN MODEL

Introduction 4.1

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid data, such as images. They are particularly well-suited for image recognition tasks due to their ability to automatically and adaptively learn spatial hierarchies of features from input images. Here's an introduction to the key concepts and components of CNNs:

1) Convolutional Layers:

- **Convolution Operation**: Convolutional layers apply a set of filters (also called kernels) to the input image. Each filter slides (or convolves) across the image, computing dot products between the filter and local patches of the image. This operation produces a set of feature maps that represent various features of the input.
- **Activation Functions**: After convolution, an activation function (commonly ReLU - Rectified Linear Unit) is applied to introduce non-linearity into the model, allowing it to learn more complex patterns.

2) Pooling Layers:

- **Max Pooling**: The most common type, which downsamples the feature maps by taking the maximum value in each patch. This reduces the spatial dimensions (height and width) of the feature maps, retaining the most significant features.
- **Average Pooling**: Similar to max pooling, but instead of taking the maximum value, it takes the average value in each patch.

3) Fully Connected Layers:

After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. These layers are similar to traditional neural networks where each neuron is connected to every neuron in the previous layer.

Features 4.2

1. Local Connectivity

- **Convolutional Layers:** Each neuron in a convolutional layer is connected only to a small region of the input volume. This local connectivity ensures that the model learns spatial hierarchies of features and significantly reduces the number of parameters compared to fully connected networks.

2. Shared Weights (Filters)

- **Parameter Sharing:** Instead of learning a separate set of weights for every location, CNNs use the same set of weights (filters) across different parts of the input. This means the same filter is applied to different patches of the input image, allowing the network to detect the same feature in different locations.
- **Feature Maps:** Applying the same filter across the image produces a feature map, which highlights the presence of a specific feature (like an edge or texture) at various locations in the input.

3. Translation Invariance

- **Pooling Layers:** By reducing the spatial dimensions of feature maps, pooling layers make the network more robust to translations of the input image. This means that the network can recognize objects even if they are shifted slightly within the image.

4. Hierarchical Feature Learning

- **Layer-by-Layer Processing:** Lower layers in the network learn to detect simple features (such as edges and textures), while higher layers combine these features to detect more complex patterns (such as shapes and objects). This hierarchical learning mirrors the way the human visual system processes images.

5. Dimensionality Reduction

- **Pooling and Striding:** Pooling layers and strides (steps by which filters move across the input) help reduce the dimensionality of the feature maps. This not only reduces computational complexity but also helps prevent overfitting by focusing on the most salient features.

6. Sparse Interactions

- **Receptive Fields:** The convolutional and pooling operations involve only a small subset of neurons from the previous layer, leading to sparse interactions.

7. Parameter Efficiency

- **Reduced Number of Parameters:** Due to local connectivity and shared weights, CNNs typically have far fewer parameters than fully connected networks with the same number of hidden units, making them easier to train and less prone to overfitting.

8. End-to-End Learning

- **Automatic Feature Extraction**: CNNs can automatically learn and optimize feature representations directly from raw input images, eliminating the need for manual feature engineering.

Applications 4.3

1. Image Classification

- **Object Recognition:** CNNs are widely used to classify images into predefined categories. For example, they can classify images of animals, vehicles, or everyday objects into categories like cats, dogs, cars, and airplanes.
- **Scene Understanding:** Beyond individual objects, CNNs can categorize entire scenes, such as identifying whether an image depicts a beach, forest, or cityscape.

2. Object Detection

- **Bounding Box Prediction:** CNNs can identify and localize objects within an image by drawing bounding boxes around them. Techniques like Faster R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector) are popular for object detection tasks.
- **Real-Time Detection:** Applications in surveillance, autonomous vehicles, and augmented reality rely on real-time object detection to identify and track objects dynamically.

3. Image Segmentation

- **Semantic Segmentation:** This involves classifying each pixel in an image to understand the object class to which it belongs. Techniques like Fully Convolutional Networks (FCNs) and U-Net are commonly used for tasks like medical image analysis and autonomous driving.
- **Instance Segmentation:** A more advanced form of segmentation where individual instances of objects are separated and classified within an image, as seen in models like Mask R-CNN.

4. Face Recognition and Verification

- **Biometric Security:** CNNs are used in face recognition systems for identity verification, access control, and security applications. They can match faces in real-time with high accuracy.
- **Social Media:** Platforms like Facebook and Google Photos use CNNs to tag and organize photos based on recognized faces.

5. Medical Image Analysis

- **Disease Detection:** CNNs assist in detecting and diagnosing diseases from medical images like X-rays, MRIs, and CT scans. They are used in identifying conditions such as tumors, fractures, and retinal diseases.
- **Segmentation and Classification:** CNNs are applied to segment organs and tissues in medical images and classify abnormalities.

CHAPTER 5

PROJECT

Description 5.1

This project aims to create an interactive system that allows users to control YouTube using hand gestures detected in real-time via a webcam. By leveraging the mediapipe library for efficient hand detection and a Convolutional Neural Network (CNN) for accurate gesture classification, the system can recognize specific hand gestures and map them to YouTube control actions such as adjusting the volume, pausing/playing videos, and skipping forward or backward.

The pyautogui library is utilized to simulate the required keyboard inputs based on the detected gestures, enabling hands-free control of the YouTube player.

This innovative approach combines advanced machine learning techniques with practical applications in human-computer interaction.

Some Key Features:

1) Real-Time Hand Detection: Utilizes the mediapipe library to accurately detect and track hand landmarks in real-time from webcam input.

2) Gesture Classification: Employs a Convolutional Neural Network (CNN) model to classify hand gestures into predefined categories such as "volume up," "volume down," "play/pause," and more.

3) YouTube Control Integration: Uses the pyautogui library to send keyboard commands to the YouTube player based on recognized gestures, facilitating seamless interaction.

4) Dynamic Hand Image Processing: Crops and resizes detected hand regions to a fixed size to ensure consistent input to the CNN model.

5) Control Delay Mechanism: Implements a delay mechanism to prevent multiple commands from being sent in quick succession, enhancing the usability and reliability of the system.

Technology Stack 5.2

Python: Programming language used for development.

OpenCV: Library for computer vision tasks such as image and video processing.

Mediapipe: Framework for building multimodal (eg. video, audio) applied ML pipelines.

TensorFlow: Deep learning framework used for building and training the hand gesture recognition model.

Keras: High-level neural networks API, used with TensorFlow for model training and evaluation.

PyAutoGUI: Python module for programmatically controlling the mouse and keyboard.

NumPy: Library for numerical computing, used for array manipulation.

Matplotlib: Library for creating static, animated, and interactive visualizations in Python.

SciPy: Library for scientific computing and technical computing.

Code Implementation 5.3

```
import os
import cv2
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import math
import time
import pyautogui
import mediapipe as mp
```

(importing necessary libraries)

```
def load_images_from_folder(folder):
    images = []
    labels = []
    gesture_classes = {gesture_class: idx for idx, gesture_class in
enumerate(os.listdir(folder))}
    for gesture_class, idx in gesture_classes.items():
        class_folder = os.path.join(folder, gesture_class)
        if os.path.isdir(class_folder):
            for filename in os.listdir(class_folder):
                img_path = os.path.join(class_folder, filename)
                img = cv2.imread(img_path)
                if img is not None:
                    img = cv2.resize(img, (32, 32))
                    images.append(img)
                    labels.append(idx)
    return np.array(images), np.array(labels), gesture_classes
```

(loading data)

```
try:
    # Set the path to your project folder using a raw string
    folder = "data"

    # Load images from folders
    images, labels, gesture_classes = load_images_from_folder(folder)
    print(f"Loaded {len(images)} images.")

    # Normalize pixel values
    images = images / 255.0

    # Convert labels to one-hot encoding
    labels = to_categorical(labels, num_classes=len(gesture_classes))

    # Split dataset into training and validation sets
    train_images, val_images, train_labels, val_labels =
train_test_split(images, labels, test_size=0.2, random_state=42)
    print(f"Split into {len(train_images)} training and {len(val_images)}
validation samples.")

    # Define model architecture
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(len(gesture_classes), activation='softmax')
    ])

    # Compile model
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    print("Model compiled successfully.")
```

(training model)

```
# Train model
    model.fit(train_images, train_labels, epochs=10,
validation_data=(val_images, val_labels))
    print("Model trained successfully.")

    # Save model and labels
    os.makedirs('Model', exist_ok=True)
    model.save('Model/keras_model.h5')
    with open('Model/labels.txt', 'w') as f:
        for gesture_class, idx in gesture_classes.items():
            f.write(f"{idx} {gesture_class}\n")
    print("Model and labels saved successfully.")

    # Evaluate model
    test_loss, test_acc = model.evaluate(val_images, val_labels)
    print('Test accuracy:', test_acc)

except Exception as e:
    print(f"An error occurred: {e}")
```

(training model)

- Hand gesture detection:

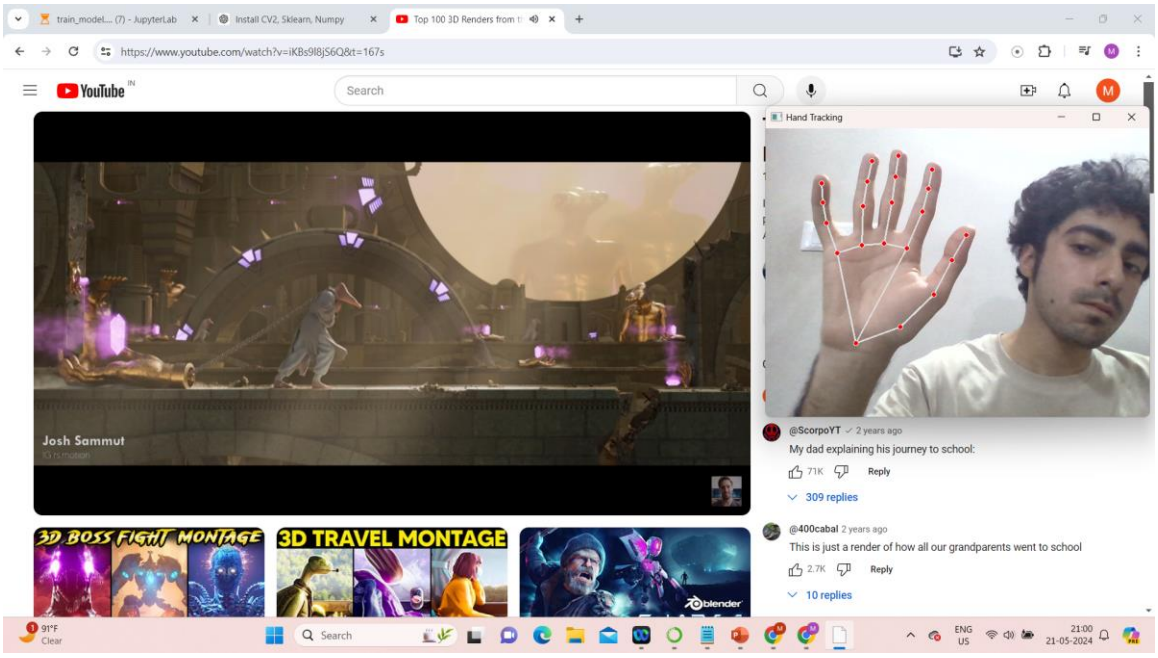
```
cap = cv2.VideoCapture(0)

# Check if the webcam is opened correctly
if not cap.isOpened():
    print("Error: Could not open video capture device.")
    exit()

# Initialize hand detector
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

try:
    while True:
        success, img = cap.read()
        if not success:
            print("Failed to capture image")
```

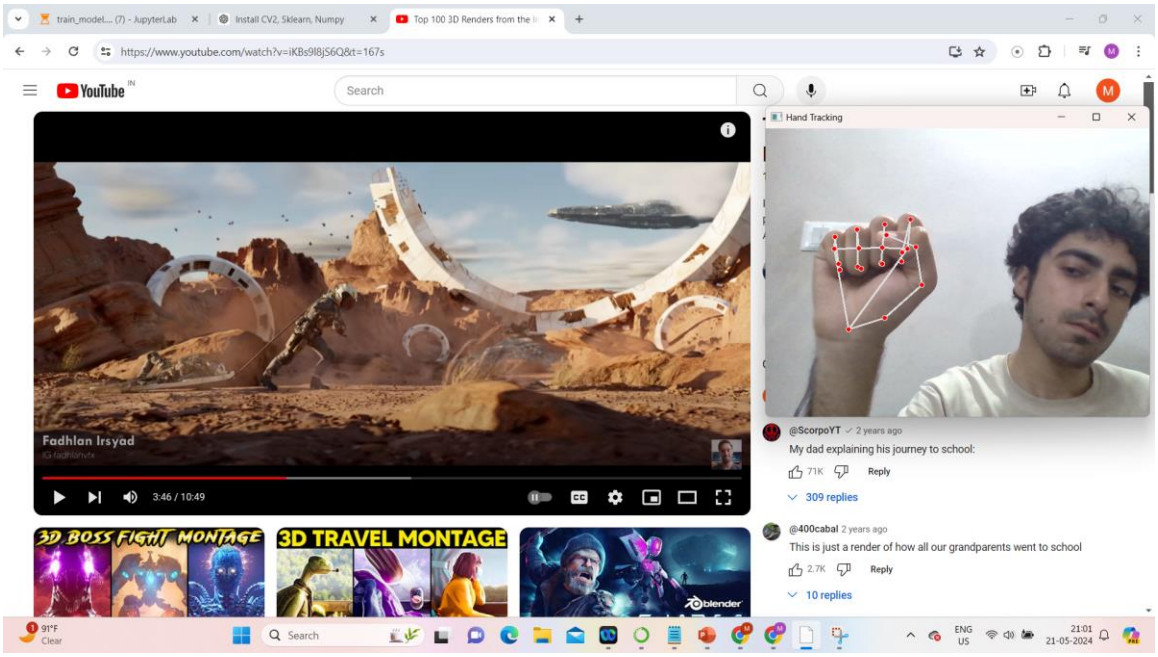
```
continue

        imgOutput = img.copy()
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        result = hands.process(imgRGB)

        if result.multi_hand_landmarks:
            for hand_landmarks in result.multi_hand_landmarks:
                mp_draw.draw_landmarks(imgOutput, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

        cv2.imshow("Hand Tracking", imgOutput)

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

finally:
    cap.release()
    cv2.destroyAllWindows()
detector = HandDetector(maxHands=1)
classifier = Classifier('Model/keras_model.h5', 'Model/labels.txt')

offset = 20
imgSize = 300

counter = 0

labels = ["free_hand","right","left", "v_up","v_down","max","min","stop"]

control_delay = 5  # Delay in seconds before performing another control
action
last_action_time = time.time() - control_delay
while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
   if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
```

```
imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset: y + h + offset, x - offset: x + w +
offset]

        if imgCrop.size == 0:
            print("Empty image crop. Skipping frame.")
            continue

        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,
draw=True)
            print(prediction, ":", index)
 else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,
draw=True)
            print(prediction, ":", index)

        cv2.rectangle(imgOutput, (x - offset, y - offset - 50), (x -
offset + 90, y - offset - 50 + 50), (255, 0, 255),
                      cv2.FILLED)
        cv2.putText(imgOutput, labels[index], (x, y - 20),
cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w +
offset, y + h + offset), (255, 0, 255), 4)

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)
```

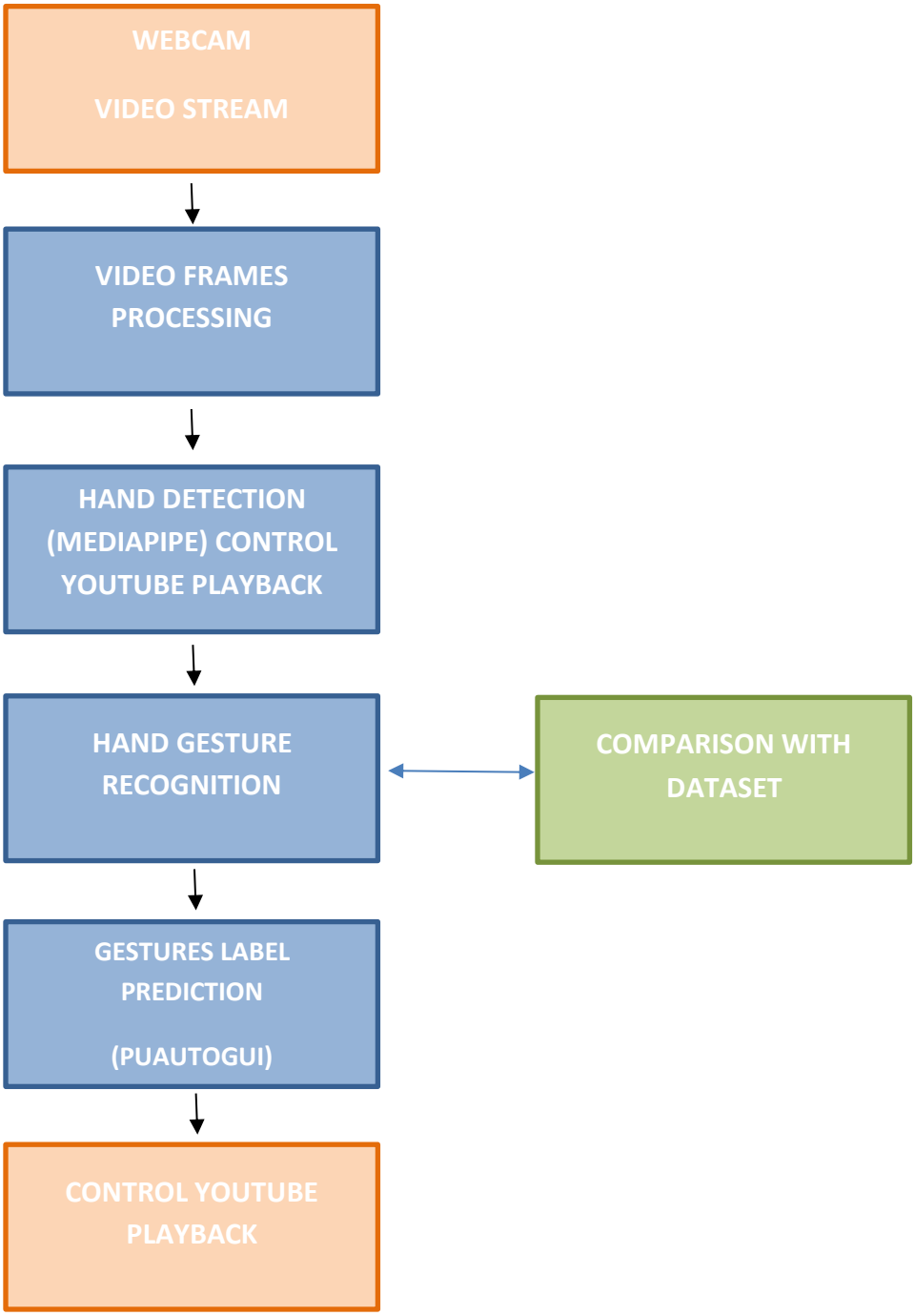
```
# Perform YouTube controls based on the predicted label
        current_time = time.time()

        if labels[index] == "v_up" :
            pyautogui.press("up")  # Press up arrow key for volume up
            last_action_time = current_time
        elif labels[index] == "v_down":
            pyautogui.press("down")  # Press down arrow key for volume
down
            last_action_time = current_time
        elif labels[index] == "free_hand":
            pass  # Do nothing for the "free_hand" gesture
        elif labels[index] == "stop" and current_time - last_action_time
>= control_delay:
            pyautogui.press("space")  # Press spacebar to pause/play
            last_action_time = current_time
        elif labels[index] == "max" and current_time - last_action_time >=
control_delay:
            pyautogui.press("f")  # Press f to enter full screen
            last_action_time = current_time
        elif labels[index] == "min" and current_time - last_action_time >=
control_delay:
            pyautogui.press("esc")  # Press f to enter full screen
            last_action_time = current_time
        elif labels[index] == "right" and current_time - last_action_time
>= control_delay:
            pyautogui.press("right")  # Press Right Arrow to skip 5 sec
            last_action_time = current_time
        elif labels[index] == "left" and current_time - last_action_time
>= control_delay:
            pyautogui.press("left")  # Press Left Arrow to revind 5 sec
            last_action_time = current_time
cv2.imshow("Image", imgOutput)
    key = cv2.waitKey(1)
    if key == ord('q'):
        break
```

Outputs 5.4


https://www.youtube.com/watch?v=iKBs9l8jS6Q&t=167s
YouTube
Search
Josh Sammut
3D BOSS FIGHT MONTAGE
3D TRAVEL MONTAGE
Hand Tracking
@ScorpoYT 2 years ago
My dad explaining his journey to school:
71K
Reply
309 replies
@400cabal 2 years ago
This is just a render of how all our grandparents went to school
2.7K
Reply
10 replies


https://www.youtube.com/watch?v=iKBs9l8jS6Q&t=167s
YouTube
Search
Fadhlan Irsyad
3:46 / 10:49
3D BOSS FIGHT MONTAGE
3D TRAVEL MONTAGE
Hand Tracking
@ScorpoYT 2 years ago
My dad explaining his journey to school:
71K
Reply
309 replies
@400cabal 2 years ago
This is just a render of how all our grandparents went to school
2.7K
Reply
10 replies

Workflow 5.4


WEBCAM
VIDEO STREAM
VIDEO FRAMES PROCESSING
HAND DETECTION (MEDIAPIPE) CONTROL YOUTUBE PLAYBACK
HAND GESTURE RECOGNITION
COMPARISON WITH DATASET
GESTURES LABEL PREDICTION
(PUAUTOGUI)
CONTROL YOUTUBE PLAYBACK

CONCLUSIONS AND FUTURE SCOPE

This project successfully demonstrates a real-time hand gesture recognition system integrated with YouTube control using a combination of Mediapipe for hand detection and a Convolutional Neural Network (CNN) for gesture recognition. The implementation effectively captures live video streams, processes the frames to detect hand landmarks, and classifies the gestures to perform specific actions on YouTube, such as play, pause, volume control, and navigating the video.

Future Scope

While the project has achieved its primary objectives, there are several areas for improvement and expansion:

- **Expanded Gesture Set:** Increase the number of recognizable gestures to support more complex interactions and additional control features.

- **Model Improvement:** Further train and refine the CNN model with a larger and more diverse dataset to improve accuracy and robustness.

REFRENCES

- Jupyter Lab official website: **Jupyter**
- Referenced article: Hand tracking with Mediapipe
- OpenCV Documentation: OpenCV
- PyAutoGui Documentation: PyAutoGUI
- "Hand Gesture Recognition Based on Deep Learning and Image Preprocessing" - T. Ahmed, M. A. Uddin, and M. Hasan
- "Vision-Based Hand Gesture Recognition Using Deep Learning for Human-Computer Interaction" - A. C. K. de Silva, C. B. Dharmasena, and A. S. Perera
- "Hand Gesture Recognition Using Deep Learning and Mediapipe Framework" - K. Wadhwa, A. Gupta, and P. Verma
- "A Real-Time Hand Gesture Recognition System Using Convolutional Neural Networks" - J. Kim and S. Kim
- "Hand Gesture Recognition: A Review" - M. Mohandes, M. Deriche, and J. Liu