Hand Gesture Recognition Documentation

1- Project Planning

1.1. Problem Definition

In recent years, advancements in computer vision and artificial intelligence have opened new possibilities for natural and intuitive human–computer interaction. Despite these developments, traditional interaction methods such as keyboards, mice, and touchscreens continue to limit accessibility and fluidity in digital communication. To address this gap, there is a growing need for systems capable of interpreting human gestures accurately and efficiently.

The problem addressed in this project is the development of an intelligent system that can recognize and interpret static hand gestures in real time using a standard webcam. The system aims to enable users to interact with a computer through predefined gestures, such as a “thumbs up” or “palm open,” eliminating the need for physical input devices.

This project focuses on designing and implementing a gesture recognition model utilizing **Python, TensorFlow, MediaPipe,** and **OpenCV,** with **Streamlit** employed for the graphical user interface. The model will be trained to detect and classify static hand gestures based on image input captured from a webcam. The goal is to achieve high accuracy, real-time responsiveness, and an intuitive user experience, demonstrating how AI-based gesture recognition can enhance accessibility and human–computer interaction.

1.2. Objectives

The primary objective of this project is to develop an artificial intelligence–based application capable of recognizing static hand gestures in real time to facilitate natural human–computer interaction.

To achieve this overarching goal, the project pursues the following specific objectives:

1. **Design and implement** a system that captures hand gestures using a standard webcam and processes them in real time.
2. **Develop and train** a deep learning model using **TensorFlow** and **MediaPipe** to accurately classify predefined static hand gestures such as “thumbs up” or “palm open.”
3. **Integrate computer vision techniques** with **OpenCV** to enhance image acquisition, preprocessing, and hand landmark detection.
4. **Build an interactive user interface** using **Streamlit** to allow users to visualize gesture recognition results and interact with the system intuitively.
5. **Evaluate system performance** in terms of accuracy, speed, and usability to ensure practical and reliable operation on standard laptop hardware.
6. **Demonstrate the potential** of gesture-based interfaces as an accessible and efficient alternative to traditional input devices for human–computer interaction.

1.3. Assemble Tools and Technologies

Real-Time Hand Gesture Recognition System was built using a combination of AI, computer vision, and interface technologies, each chosen to ensure efficiency, reliability, and a real-time performance.

The foundation of the project is **Python**, a programming language widely used in AI development. Its libraries allowed us to handle tasks ranging from image processing and model training to real-time visualization and deployment.

During the development and training phase, we relied heavily on **TensorFlow** and **Keras** for building, training, and saving the deep learning model. These frameworks provided the essential layers, callbacks, and optimization tools (like EarlyStopping and ModelCheckpoint) that helped improve model accuracy and prevent overfitting. The model was trained using fully connected layers defined through Sequential, with preprocessing, augmentation, and segmentation managed using NumPy and OpenCV.

The model development and training were carried out both locally and on Google Colab. Colab provided GPU acceleration, which greatly reduced training time and allowed for experimentation with different architectures and hyperparameters, while local execution allowed better control over testing, debugging, and integration with other modules such as the real-time application and interface.

For data handling and analysis, **Pandas** and **NumPy** played a major role in managing datasets and performing numerical computations, while **Matplotlib** and **Seaborn** were used to visualize model performance, loss curves, and confusion matrices during evaluation.

Several **supporting tools and libraries** enhanced the project's workflow and efficiency:

- **Pickle:** Used to store and load the trained model and label encoder, ensuring consistent gesture label decoding during inference.
- **Scikit-learn:** Utilized for evaluating model performance using metrics such as classification reports and confusion matrices.
- **Random** and **OS:** Assisted in dataset management, file handling, and random shuffling during data preparation.

In the real-time application phase, **MediaPipe** was the core library for detecting and tracking hand landmarks. It provided accurate coordinates of each hand joint, which were then passed into the trained neural network for classification. **OpenCV** was responsible for capturing video frames, flipping and converting them, overlaying bounding boxes, and rendering the final real-time predictions on the video feed.

To make the system more interactive, **Streamlit** was used to develop a simple web-based interface. users can start and stop gesture recognition, view live camera feed directly on the browser, and interact with the model without running any code manually.

Together, these tools form a development pipeline:

- **Data Collection and Training:** handled using TensorFlow, Keras, Pandas, NumPy, and visualization libraries.
- **Real-Time Detection and Inference:** powered by MediaPipe and OpenCV.
- **User Interaction:** managed through Streamlit for live testing and demonstration.

This helps ensure a smooth operation from data preparation and model training to real-time recognition, creating a responsive and accurate hand gesture recognition system.

1.4. Determining Project Milestones

Determining milestones were set to keep the project organized and ensure continuous progress. Each phase has a goal to finish, from data preparation to real-time deployment and monitoring, allowing to build, test, and refine the system efficiently at every stage.

The project was organized into four main development milestones:

1. **Data Collection and Preprocessing**: focused on gathering gesture data, cleaning it, and preparing it for model training through feature extraction and exploration.
2. **Model Development and Training**: dedicated to designing, training, and fine-tuning the deep learning architecture for accurate gesture classification.
3. **Real-Time Application and Deployment**: focused on integrating the trained model into a live recognition system, enabling real-time interaction through camera input and a user-friendly interface.
4. **MLOps Implementation and Model Monitoring**: aimed at ensuring system stability and continuous improvement for future updates.

1.4.1 Data Collection and Preprocessing

The first phase of the project focused on building a high-quality dataset to train the hand gesture recognition model. Initially, we explored several publicly available datasets but found that most of them were either limited in gesture variety, inconsistent in lighting and angles, or did not match our real-time recognition requirements.

To overcome this challenge, we developed a **custom data collection tool** that allowed us to capture various hand gestures directly from team members. This approach ensured consistency in data quality, background conditions, and gesture diversity, giving us a dataset that closely reflected real-world usage scenarios.

After collecting the data, we conducted an initial **exploratory data analysis** (EDA) to understand its distribution. Using **histograms and visual plots**, we examined the number of samples per class to confirm balance across gesture categories and detect any inconsistencies.

Next, we focused on improving the dataset's quality through **image preprocessing techniques**. A color-based **segmentation function** was developed to isolate the hand region from the background, enhancing the clarity of gesture shapes and minimizing noise.

Following segmentation, we applied **normalization** to scale pixel values and maintain uniformity across images, ensuring better model convergence during training. To further strengthen the model's robustness, we also implemented **data augmentation**, generating slightly modified versions of existing images by applying transformations such as rotation, flipping, and brightness adjustments.

These steps ensured that the dataset was clean, balanced, and diverse, forming a strong foundation for the model training phase.

1.4.2 Model Development and Training

In this phase, we focused on building and training the deep learning architecture for gesture classification. Using MediaPipe, we extracted detailed hand landmarks, including joint positions and angles, which served as the primary input features for the model. These extracted values were then stored in files to streamline the workflow, allowing us to reuse the processed data without repeating the extraction process each time.

The model was developed using TensorFlow and Keras, where we designed a fully connected neural network capable of learning the spatial relationships between hand joints. We incorporated techniques such as batch normalization and dropout to improve generalization and prevent overfitting.

Once trained, the model demonstrated strong performance in recognizing various gestures, forming the backbone of the real-time recognition system built in the following phase.

1.4.3 Real-Time Gesture Recognition and Deployment

In this stage, we integrated the trained model into a real-time recognition system using OpenCV and MediaPipe. The system captures video through the webcam, processes it frame by frame, and sends each frame to the trained model for prediction. The predicted gesture and its corresponding confidence percentage are then displayed directly on the screen, creating a smooth, interactive experience.

To make the system more user-friendly, we built a Streamlit interface that allows users to launch the real-time recognition directly from a web app without running code manually. This simplified the interaction and made the project easier to test and demonstrate.

Additionally, this phase extended into practical applications, such as using hand gestures to control computer actions, for instance, moving the mouse cursor, flipping presentation slides, or performing specific system commands. This integration will demonstrate how the model can be applied in real-world scenarios, making human-computer interaction more natural and touch-free.

1.4.4 MLOps Implementation and Model Monitoring

The final phase focuses on implementing MLOps (Machine Learning Operations) to ensure that the system remains reliable, scalable, and easy to maintain after deployment. This phase plays a vital role in transitioning the model from a working prototype to a production-ready application capable of continuous improvement.

Through MLOps, the system automates processes such as model retraining, version control, and continuous integration and deployment (CI/CD). These practices allow the model to adapt as new gesture data becomes available, improving performance over time with minimal manual intervention. Additionally, monitoring tools are used to track model accuracy, latency, and overall system health in real-world conditions.

By adding MLOps practices, the gesture recognition system maintains consistency, reduces deployment risks, and supports future updates seamlessly, ensuring that it remains efficient, scalable, and aligned with evolving user needs.

2. Data Analysis & Visualization

2.1. Class Distribution Analysis

To ensure the model's robustness and ability to generalize, we performed a detailed analysis of the dataset's class distribution across the training, validation, and test sets.

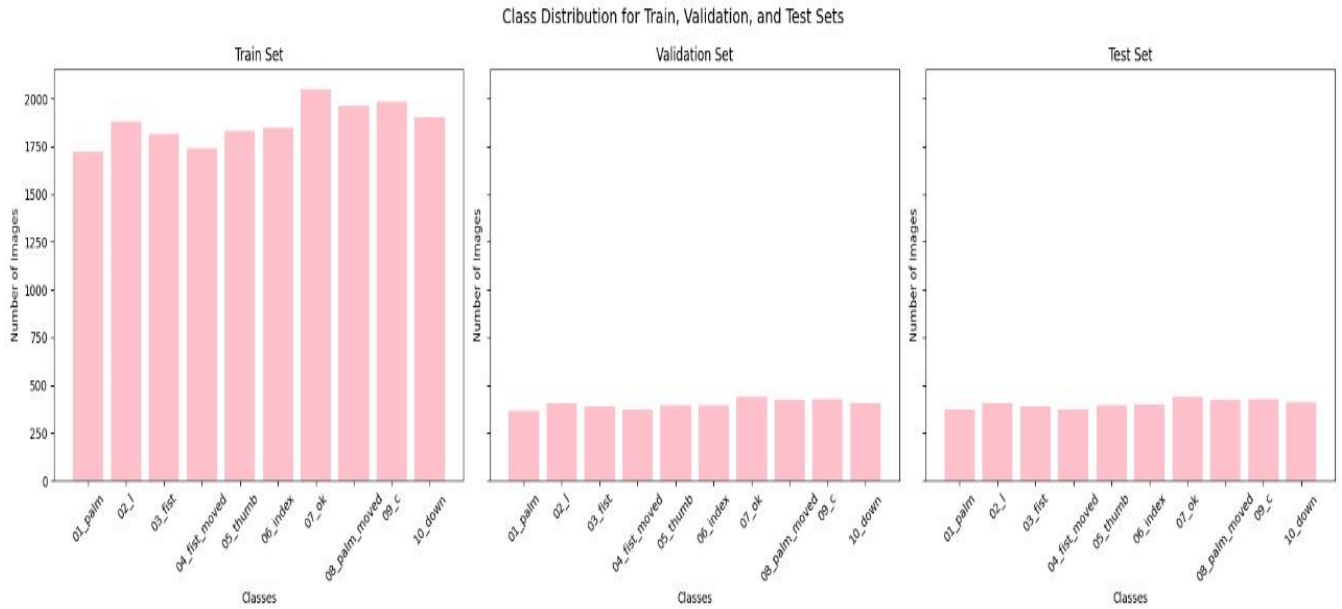


Figure 2.1: Class Distribution across Train, Validation, and Test Sets.

Analysis: As illustrated in Figure 2.1, the dataset exhibits a relatively balanced distribution among the 10 gesture classes (e.g., '01_palm', '02_l', '03_fist', etc.). The training set contains the largest portion of data, approximately 1700-2000 images per class, providing ample examples for the deep learning model to learn varied features. The validation and test sets also show consistent distribution, ensuring that performance evaluation metrics are not biased towards any specific class. This balance is crucial for preventing the model from developing a bias towards majority classes during training.

2.2. Pixel Intensity Distribution

We analyzed the pixel intensity distribution to understand the lighting and contrast characteristics of the dataset.

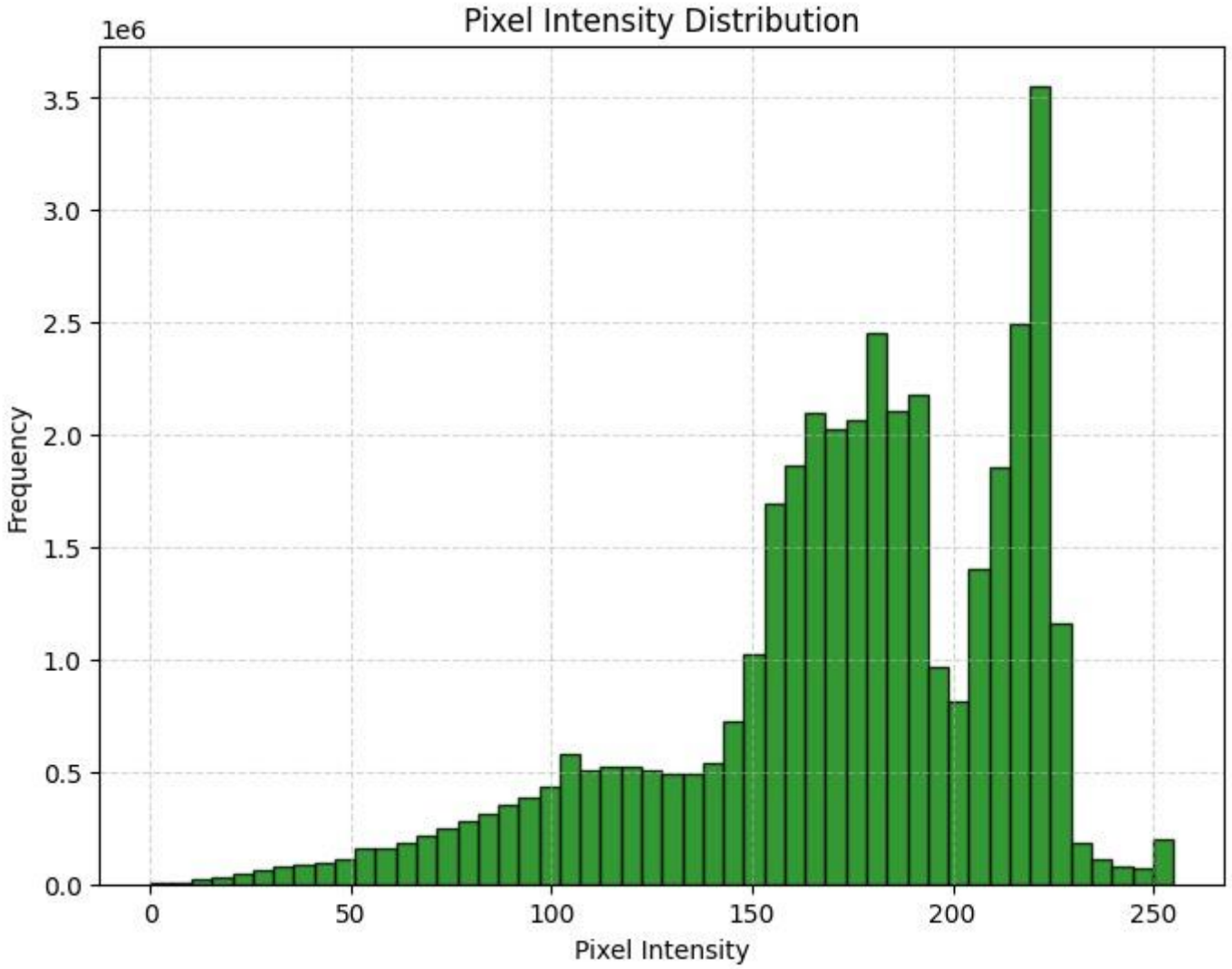


Figure 2.2: Pixel Intensity Distribution Histogram.

Analysis: Figure 2.2 displays the frequency of pixel intensities across the dataset. The distribution is multi-modal, with significant peaks in the higher intensity range (around 220-230), indicating areas of brightness or skin tones against the background. The spread across the 150-200 range suggests varying lighting conditions. This insight confirms the necessity of the normalization step (scaling pixel values to [0, 1]) implemented during preprocessing to standardize the input data for the neural network, facilitating faster convergence.

3. System Design & Architecture

3.1. System Architecture Description

The system follows a streamlined pipeline architecture designed for efficient real-time processing. The data flow is structured into the following stages:

1. **Input Layer:** Captures live video frames via a standard Webcam using the OpenCV library.
2. **Feature Extraction:** Each captured frame is processed by **MediaPipe Hands** to extract 21 3D landmarks (x, y, z) for each detected hand.
3. **Preprocessing Module:**
    - **Flattening:** The 3D coordinates are converted into a 1D array.
    - **Normalization:** Landmarks are adjusted relative to the wrist (Origin Index 0) and scaled based on the maximum distance between points to ensure the model is invariant to hand distance and position within the frame.
4. **Inference Engine:** The processed vector is fed into the trained **TensorFlow/Keras** Deep Neural Network (DNN)
5. **Output Layer:** The model outputs a probability distribution using a Softmax activation function, which is then decoded into specific gesture labels (e.g., "Fist", "Palm", "Thumb").
6. **Application Layer:** The final result is displayed on the **Streamlit** user interface with associated confidence scores

3.2. User Interface (UI) Design

Based on the developed application (main.py), the user interface is designed with a "Wide Layout" for optimal visibility. The UI components include:

- **Header:** A title "Hand Gesture Recognition" displayed with a gradient style.
- **Two-Column Layout:**
    - **Left Column (Video Feed):** Displays the live camera input with a visual overlay card showing the recognized gesture and its status.
    - **Right Column (Predictions & Controls):**
        - **Action Buttons:** Controls to "Start Camera", "Stop Camera", and "Clear History".
        - **Prediction Slot:** Real-time display of "Right Hand" and "Left Hand" detection results with confidence percentages.
        - **Top-3 Probabilities:** A list of the top 3 possible gestures when the model's confidence is split, aiding in debugging and transparency.

4. Model Architecture & Training

4.1. Neural Network Structure

The gesture classification model is a Fully Connected Deep Neural Network (DNN) built using the Keras Sequential API. It is specifically designed to process the 63-point input vector (21 landmarks X times 3 coordinates).

Model Summary:

Layer Type	Output Shape	Parameters	Activation	Function
Input Layer (Dense)	(None, 256)	16,384	-	Receives the flattened landmark vector.
BatchNormalization	(None, 256)	1,024	-	Normalizes inputs to stabilize training.
Dense	(None, 256)	65,792	ReLU	First hidden layer for feature learning.
BatchNormalization	(None, 256)	1,024	-	-
Dropout	(None, 256)	0	-	Dropout rate of 0.3 to prevent overfitting.
Dense	(None, 128)	32,896	ReLU	Second hidden layer.
BatchNormalization	(None, 128)	512	-	-
Dropout	(None, 128)	0	-	Dropout rate of 0.3.
Dense	(None, 64)	8,256	ReLU	Third hidden layer.
BatchNormalization	(None, 64)	256	-	-
Dropout	(None, 64)	0	-	Dropout rate of 0.2.
Output Layer (Dense)	(None, 10)	650	Softmax	Outputs probabilities for the 10 gesture classes.

- **Total Parameters:** 126,794 (Trainable: 125,386)
- **Optimizer:** Adam
- **Loss Function:** Categorical Crossentropy

4.2. Training Performance

The model was trained over 50 epochs. The training logs indicate:

- **Training Accuracy:** Reached approximately **99.8%**.
- **Validation Accuracy:** Consistently reached **99.8%**.
- **Loss:** Decreased significantly from ~1.29 in the first epoch to ~0.01 by the final epoch, indicating excellent convergence without significant overfitting, largely due to the use of Dropout and Batch Normalization layers

5. Visualizations & Evaluation

5.1. Performance Metrics

To evaluate the model's performance, Matplotlib and Seaborn libraries were used to generate visual insights:

1. **Accuracy Plot:** Visualizes the trajectory of Training vs. Validation accuracy, showing a steady increase over the 50 epochs.
2. **Loss Plot:** Demonstrates the Training vs. Validation loss converging to near zero, confirming the model's stability.
3. **Confusion Matrix:** Generated using sklearn.metrics.confusion_matrix, this matrix provides a breakdown of classification performance per gesture class, highlighting any specific gestures that may be confused with one another.

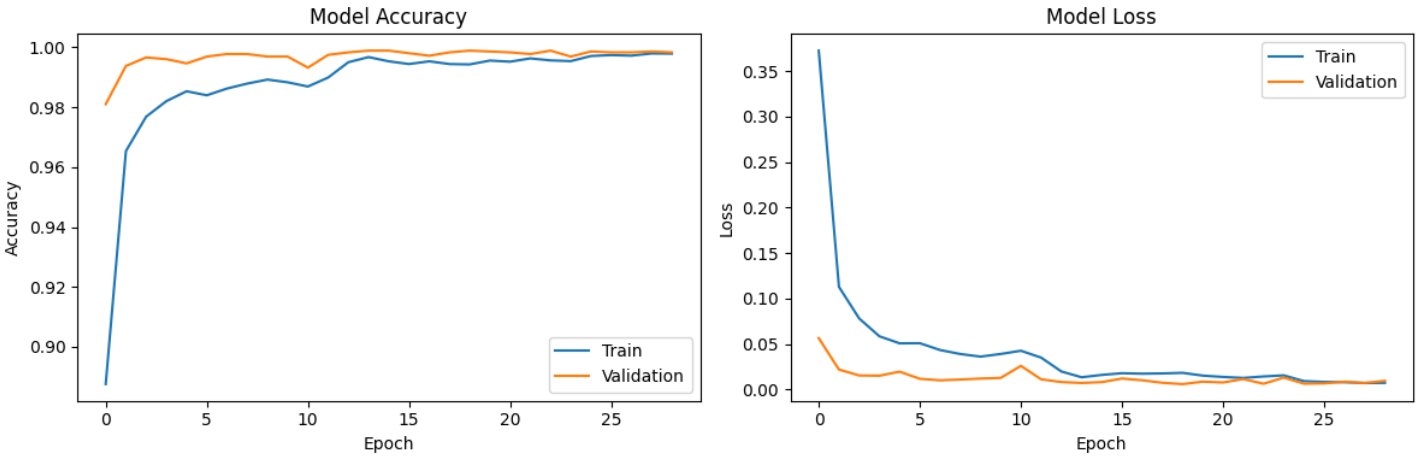


Figure 5.1.1: Model Accuracy vs Model Loss.

Analysis: As shown in Figure 5.1.1, the model demonstrates strong learning behavior across epochs. The training accuracy steadily increases and approaches nearly **99%**, while the validation accuracy closely follows, showing minimal divergence. This indicates that the model is generalizing well and not overfitting to the training data. Similarly, the loss curves show a rapid decrease during the initial epochs, with both training and validation loss converging toward very low values. The consistent alignment between the two curves suggests stable training dynamics, effective feature learning, and a well-tuned network architecture.

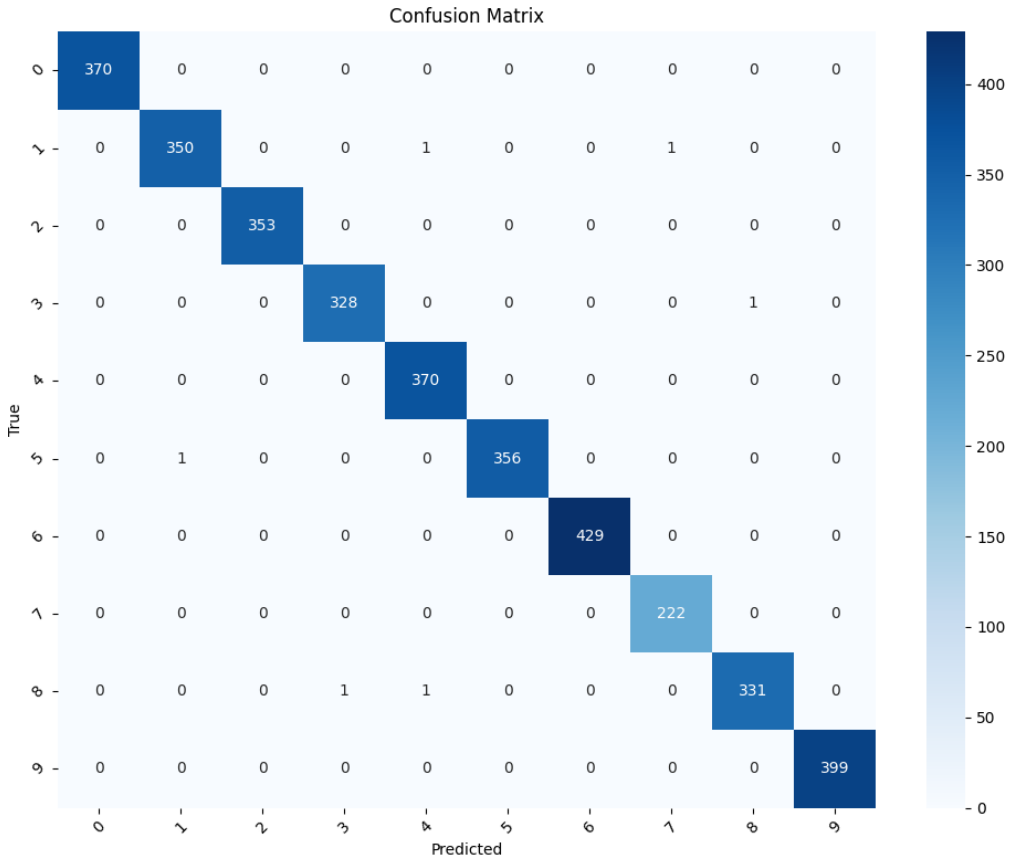


Figure 5.1.2: Confusion Matrix.

Analysis: The confusion matrix illustrates the model's performance across all gesture classes. Each class shows a very high number of correct predictions, with almost all samples aligned along the diagonal of the matrix. Misclassifications are minimal and distributed sparsely, indicating that the model is highly capable of distinguishing between the different hand gestures. This strong diagonal dominance confirms that the trained CNN is effectively capturing the visual patterns and features that differentiate each gesture, resulting in robust classification performance across all categories.

6. Tutorials (User Manual)

How to Run the Application
The system uses Streamlit for the web interface. Follow these instructions to launch and operate the system:

1. Prerequisites:
Ensure **Python 3.10+** is installed. Install the required libraries using:

pip install tensorflow mediapipe opencv-python streamlit

2. Launching the App:
Navigate to the project directory in your terminal and run the following command:

streamlit run main.py

3. Operating the Interface:

- **Start:** Click the **"Start Camera"** button. The webcam feed will appear in the left column.
- **Interaction:** Raise your hand in front of the camera. Ensure good lighting for best results.
- **Feedback:** The recognized gesture (e.g., "Palm", "Fist", "OK") will appear overlaid on the video feed and in the "Predictions" panel on the right.
- **Reset:** To clear the prediction smoothing history, click the **"Clear History"** button.
- **Stop:** To close the camera feed and stop recognition, click **"Stop Camera"**.

4. **Running the Gesture-Control Use Case (Optional):**

In addition to the main interface, you can run an extended use case that triggers predefined actions based on recognized gestures. This mode tracks real-time predictions and performs specific operations such as scrolling, changing media (forward/backward music), controlling mouse movements, navigating to the next or previous slide in a PDF, and more, whenever specific gestures are detected.
To launch this controller, run:
python realtime_gesture_controller.py

7. Technical Report Summary

The developed **Real-Time Hand Gesture Recognition System** successfully leverages a lightweight Deep Neural Network (DNN) to classify static hand gestures with high efficiency. By integrating **MediaPipe** for robust landmark extraction, the system eliminates the need for raw image processing, making it resilient to variations in lighting and background noise.

Key Technical Achievements:

- **High Accuracy:** The model achieved a validation accuracy of **99.8%**, demonstrating superior classification capability for the 10 target gesture classes.
- **Real-Time Efficiency:** The inference pipeline is optimized with landmark normalization (max_dist scaling) and rotation invariance, allowing it to run smoothly on standard CPU hardware without the need for heavy GPU resources during inference.
- **Robust Architecture:** The inclusion of Batch Normalization and Dropout layers effectively prevented overfitting during the training cycle.
- **User-Centric Design:** The Streamlit-based interface provides a clear, responsive, and easy-to-use platform for demonstrating the model's capabilities in real-time.

8- Stakeholder Analysis

Stakeholder	Role	Interest	Influence	Expectations / Needs
DEPI Program Management	Sponsor / Supervisor	High	Medium	Ensure project success, proper use of ML methods, teamwork, and timely completion.
Mentor / Technical Instructor	Technical Advisor	High	Medium	Provide technical guidance, review model design, and support in solving challenges.
Project Team Members	Developers / Data Scientists / Dataset Creators	High	High	Collect and preprocess custom data, build and train the model, and meet DEPI project requirements.
End Users / Testers	System Users	Medium	Low	Expect a reliable, accurate, and user-friendly gesture recognition system.
Evaluation Committee	Reviewer	High	High	Evaluate project quality, innovation, and overall performance.

Summary

The success of the project relies heavily on effective teamwork between the DEPI mentors and the project team.

Since the dataset is self-collected and prepared by the developers, data quality and preprocessing play a key role in the model's performance.

Regular communication, testing, and model refinement are essential for ensuring accurate real-time gesture recognition and a successful project outcome.

9- Database Design

The Real-Time Hand Gesture Recognition System works by capturing live video input, recognizing hand gestures, and responding immediately with specific actions, such as navigating slides or controlling simple interface commands. The process happens instantly — the system detects the hand, identifies the gesture, and performs the corresponding function in real time.

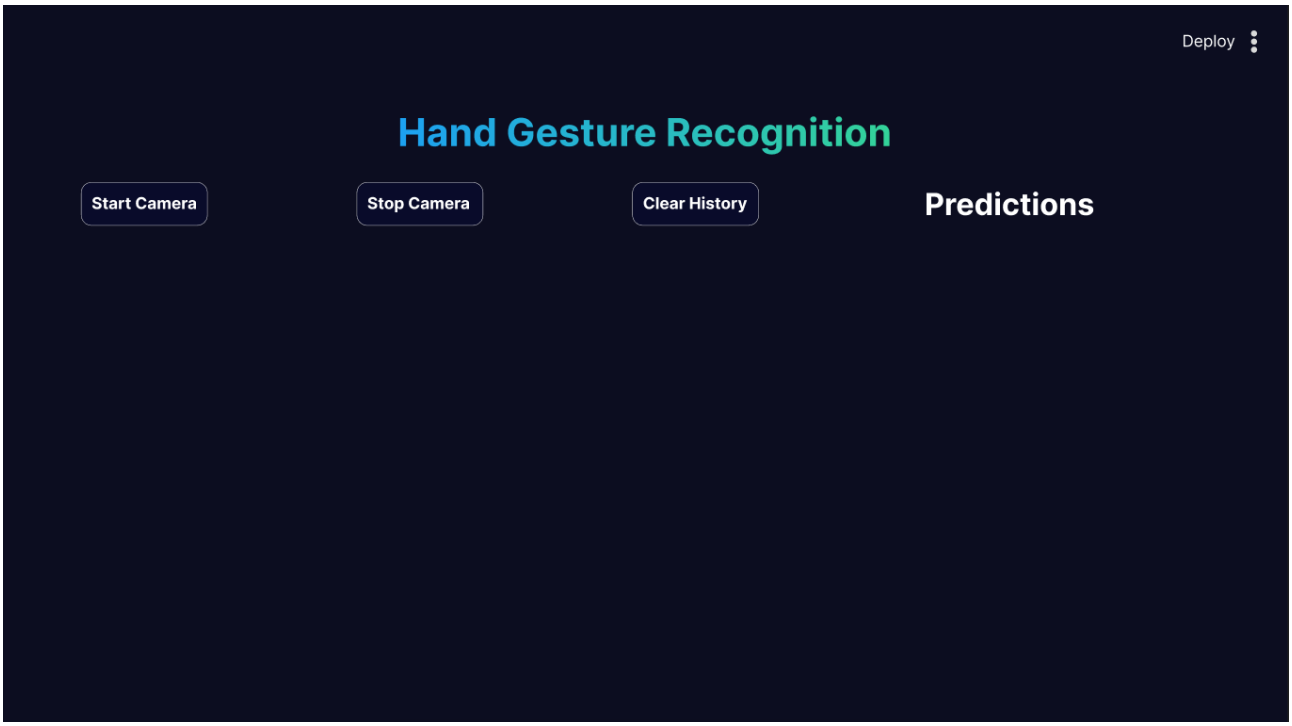
Because no user information, images, or gesture history needs to be stored, there was **no requirement for a database** in this project. All operations occur live during system execution, and once the session ends, no data is retained.

As a result, a database design was considered unnecessary, since the system's purpose is to process gestures in real time rather than manage or store data for future use.

10- UI/UX Design

To ensure the system is easy to navigate and provides a smooth user experience, We designed a clean and functional interface. The UI focuses on clarity, minimalism, and guiding the user through each stage of the gesture-recognition process, from capturing the gesture to displaying real-time predictions.
You can view the full design here: **https://www.figma.com/design/7uip0oa2NKQOEZaIwtVuZO/Depi-Graduation-Project?node-id=0-1&t=r1ZlVF7lmPS7EMqU-1**

And here are images showing the project in action:



Stop Deploy

Hand Gesture Recognition

Predictions

Right: -

Left: -

Start Camera | Stop Camera | Clear History

Stop Deploy

Hand Gesture Recognition

Predictions

Right: Right: 01_palm (97.3%)

- **01_palm -** 97.3%
- **08_palm - moved -** 2.7%
- **10_down -** 0.0%

Left: -

Start Camera | Stop Camera | Clear History

Deploy

Hand Gesture Recognition

Start Camera | Stop Camera | Clear History

Predictions

Cleared history