**Sign Language Detection Report**

**Project Description**

**Objective:**

The objective of this project is to develop a machine learning model capable of recognizing and predicting specific sign language words. The model will be designed to function within a specific time window, from 6 PM to 10 PM daily, to simulate an operational scenario. This includes creating a Graphical User Interface (GUI) for real-time detection and image upload functionality.

**Project Components:**

1. **Data Collection and Preprocessing:**
   * Use a sign language dataset (e.g., Sign Language MNIST) for training.
   * Preprocess images to standardize input size and format.
2. **Model Development:**
   * Build and train a Convolutional Neural Network (CNN) model using TensorFlow/Keras.
   * Save the trained model and weights.
3. **Model Evaluation:**
   * Evaluate model performance using accuracy, confusion matrix, and other relevant metrics.
   * Fine-tune the model based on evaluation results.
4. **GUI Development:**
   * Develop a GUI using Tkinter for image upload and real-time video processing.
   * Integrate the trained model with the GUI.
5. **Time-based Model Operation:**
   * Implement functionality to restrict model operations to between 6 PM and 10 PM.

**Model Training File**

The Jupyter Notebook (.ipynb) file should contain the entire process of loading the dataset, preprocessing data, building and training the model, evaluating performance, and saving the model.

**Contents:**

* **Data Loading and Preprocessing:** Loading the Sign Language dataset and preprocessing images.
* **Model Architecture:** Defining the CNN architecture.
* **Training:** Training the model and monitoring performance.
* **Evaluation:** Calculating accuracy, generating a confusion matrix, and other evaluation metrics.
* **Saving the Model:** Saving the model and weights for deployment.

**3. Model Weights**

If the trained model weights file is large, upload it to Google Drive and provide a shared link. The weights file allows for quicker model loading without needing to retrain.

**Link Example:**

* Download Model Weights

**4. Saved Model**

The saved model, including architecture and weights, should be uploaded to Google Drive if the file size is too large to share directly.

**Link Example:**

* Download Saved Model

**5. GUI (Graphical User Interface)**

The GUI, developed using Tkinter, allows users to interact with the model. It should provide functionalities for:

* Uploading images for prediction.
* Real-time video feed processing.
* Displaying predictions and confidence scores.
* Operating only between the hours of 6 PM and 10 PM.

**Contents:**

* **Python script (.py)** implementing the GUI and integrating the model.
* **Instructions** on how to run the GUI locally.

**6. Model Performance**

The performance of the model should be assessed and documented using the following metrics:

* **Accuracy:** The proportion of correctly predicted signs out of all predictions.
* **Confusion Matrix:** A table used to describe the performance of the classification model by comparing actual versus predicted values.

**Project Structure:**

plaintext

Copy code

SignLanguageDetection/

│

├── requirements.txt

├── model\_training.ipynb

├── gui.py

├── saved\_model/

│ └── sign\_language\_model.h5

├── weights/

│ └── sign\_language\_weights.h5

└── README.md

**README File**

Include a README file to guide users through setting up and running the project. The README should provide:

* **Installation Instructions:** How to set up the Python environment and install dependencies using requirements.txt.
* **Usage Instructions:** How to run the Jupyter Notebook, load the model, and use the GUI.
* **Links to Large Files:** Provide links to the saved model and weights if they are hosted on Google Drive.