README

Gesture-based Calculator Project

Project Overview

This project implements a gesture-based calculator using computer vision and deep learning. The system captures hand gestures via a webcam, preprocesses the images, and predicts numerical gestures to perform basic arithmetic operations.

File Structure

- **`createDataset.py`**: Captures images of hand gestures, preprocesses them, and saves them to a directory for model training.
- **`CNN.ipynb`**: Implements a Convolutional Neural Network (CNN) to train the gesture recognition model.
- **`main.py`**: Uses the trained model to predict gestures in real-time and perform calculations.

Code Workflow

`createDataset.py`

1. **Objective**: Collect and preprocess hand gesture images.
2. **Key Steps**:
   - Captures images from the webcam.
   - Focuses on a **Region of Interest (ROI)** using a bounding box.
   - Converts the ROI to HSV format and applies a mask to detect skin color.
   - Reduces noise using Gaussian blur, dilation, and erosion.
   - Thresholds the image to extract the hand's features.
   - Saves the preprocessed images every 5 frames into the `output_images` directory.
3. **Output**:
   - A set of preprocessed binary images saved for each frame.

`CNN.ipynb`

1. **Objective**: Train a deep learning model to recognize gestures.
2. **Dataset Preparation**:
   - Loads gesture images from the dataset directory (`gestures/gestures`).
   - Augments the dataset by flipping images horizontally.
   - Normalizes pixel values to a range of [0, 1].
   - Splits data into training and testing sets (75% training, 25% testing).
3. **Model Architecture**:
   - **Convolutional Layers**: Extract spatial features.
   - **MaxPooling**: Down-sample feature maps.
   - **Dense Layers**: Fully connected layers for classification.
   - **Dropout**: Prevents overfitting.
   - **Output Layer**: Softmax activation for multi-class classification.
4. **Training**:
   - Compiles the model using the Adam optimizer and categorical cross-entropy loss.
   - Saves the best-performing model as `model.h5`.
5. **Evaluation**:
   - Visualizes input gestures.
   - Outputs model performance metrics (accuracy, loss).

`main.py`

1. **Objective**: Perform real-time gesture recognition and calculations.
2. **Setup**:
   - Loads the trained model (`model.h5`).
   - Captures video frames from the webcam.
3. **Preprocessing**:
   - Identifies the ROI in the frame.
   - Applies similar preprocessing steps as `createDataset.py`.
   - Combines thresholding with Canny edge detection for better feature extraction.
4. **Prediction**:
   - Uses the CNN model to predict numerical gestures.
   - Confirms predictions after consistency over multiple frames.
5. **Calculator Logic**:
   - Maintains an array of operands for arithmetic operations.
   - Writes predicted numbers and calculations to a scrolling window (`result`).
   - Performs addition and displays the sum.
6. **User Interaction**:
   - Press **Esc** to exit.

    - Uses a scrollbar to view longer results.

Usage Instructions

1. **Dataset Creation**:
    - Run `createDataset.py`.
    - Position your hand in the ROI (green box).
    - Perform gestures, and the script will save preprocessed images in the `output_images` directory.
2. **Model Training**:
    - Organize gesture images into labeled folders inside `gestures/gestures`.
    - Run `CNN.ipynb` to train the model and generate `model.h5`.
3. **Real-time Prediction**:
    - Run `main.py`.
    - Perform gestures in the ROI, and the calculator will display predictions and sums in the result window.

Key Features

- **Gesture Recognition**: Accurately identifies hand gestures for numbers.
- **Real-time Processing**: Displays predictions and results instantly.
- **User-friendly Interface**: Highlights gestures in a green bounding box and displays results in a separate window.

System Requirements

- Python 3.8+
- Libraries:
    - `opencv-python`
    - `numpy`
    - `keras`
    - `tensorflow`
    - `matplotlib`
    - `sklearn`
- Webcam for capturing gestures.

Acknowledgments

This project uses **OpenCV** for image processing and **Keras** for deep learning. The methodology aligns with real-world gesture recognition and calculator use cases.