**Traffic Sign Detection System – Starter Guide**

**1. Project Overview**

This project implements a traffic sign detection system using a Convolutional Neural Network (CNN). It is designed to classify traffic signs based on the GTSRB dataset. The system preprocesses images, trains a model on the dataset, and visualizes predictions on a subset of the data. This guide will help you understand how to run the project, what each file does, and how to modify the code if needed.

**2. Project Directory Structure**

Your project directory is organized as follows:

/project-directory

├── dataset/ # (Optional) Store any additional datasets here

├── env/ # Virtual environment folder

├── Meta/ # Other metadata or resources related to the project

├── models/ # Directory to store saved models (e.g., traffic\_sign\_model.h5)

├── src/ # Source folder for Python scripts

├── data\_loader.py # Script for loading and preprocessing data

├── model.py # Script defining the CNN model architecture

├── train.py # Script for training the CNN model

├── visualize.py # Script for plotting training/validation metrics

├── visualize\_predictions.py # Script for visualizing actual vs predicted labels on images

├── Test/ # Folder containing testing images (if needed)

├── Train/ # Folder containing training images

├── Meta.csv # Metadata CSV

├── Test.csv # CSV file for testing image paths and labels

├── Train.csv # CSV file for training image paths and labels

├── main.py # Main script for running the project (ties everything together)

**3. Key Files and Their Functions**

**1. data\_loader.py**

* **Purpose**: This script is responsible for loading and preprocessing the dataset. It prepares the training and validation data using ImageDataGenerator with appropriate augmentations and splits.
* **Key Functions**:
  + load\_data: Loads the data from Train.csv, resizes the images, normalizes pixel values, and returns training and validation data generators.

**2. model.py**

* **Purpose**: This script defines the CNN model architecture used for traffic sign classification.
* **Key Functions**:
  + build\_model: Constructs and returns the CNN model. It includes convolutional layers, max pooling, dropout layers for regularization, and a final dense layer for classification into 43 classes (traffic signs).

**3. train.py**

* **Purpose**: This is the main training script that ties together the data loading, model building, and training process.
* **Key Steps**:
  + Loads the training and validation data using load\_data from data\_loader.py.
  + Builds the CNN model using build\_model from model.py.
  + Trains the model using model.fit and stores training history for later visualization.
  + Saves the trained model in the models/ directory for future predictions.

**4. visualize.py**

* **Purpose**: This script is used to visualize the training and validation accuracy/loss during the model training process.
* **Key Functions**:
  + plot\_history: This function takes the training history (accuracy and loss values over epochs) and plots it for both training and validation sets.

**5. visualize\_predictions.py**

* **Purpose**: This script visualizes actual vs predicted labels for a subset of validation images.
* **Key Steps**:
  + Loads the validation dataset for predictions.
  + Loads the trained model from models/traffic\_sign\_model.h5.
  + Plots a batch of images with their actual and predicted labels side by side for easy comparison.
  + Useful for testing the performance of the model on unseen data.

**6. main.py**

* **Purpose**: This is the main entry point of the project, used to bring together all the components.
* **Key Steps**:
  + Imports the training history from train.py.
  + Calls the plot\_history function from visualize.py to visualize training progress.
  + If required, you can also add functionality here to retrain the model or run predictions by importing functions from other scripts.

**7. Train.csv and Test.csv**

* **Purpose**: These CSV files contain the image file paths and corresponding class labels for the training and test datasets. The Path column refers to the image location, and ClassId refers to the label for that image.

**4. Setting Up the Project**

**Step 1: Activate the Existing Virtual Environment**

**You have already created a virtual environment named env. To activate it, follow these steps:**

1. **Activate the virtual environment:**
   * **On Windows (PowerShell):**

**bash**

**Copy code**

**.\env\Scripts\Activate**

* + **On macOS/Linux:**

**bash**

**Copy code**

**source env/bin/activate**

**5. Running the Project**

**Step 1: Train the Model**

You can train the model using the train.py script. This will load the data, build the model, and start training. After training, the model will be saved to the models/ directory.

python src/train.py

**Step 2: Visualize Training Progress**

Once the model is trained, you can visualize the training and validation accuracy/loss using the main.py script.

python main.py

**Step 3: Visualize Predictions**

After training the model, you can use the visualize\_predictions.py script to visualize how well the model performs by comparing actual vs predicted labels on a batch of validation images.

python src/visualize\_predictions.py

**6. Modifying the Project**

**Modifying the Model:**

* To change the architecture, modify the build\_model function in model.py.

**Modifying Data Augmentation:**

* To change or add data augmentation techniques, modify the ImageDataGenerator in data\_loader.py. You can adjust the rotation\_range, zoom\_range, and other parameters for more advanced augmentation.

**7. Troubleshooting**

* **Model Overfitting**: If the validation accuracy is much lower than the training accuracy, try increasing the dropout rate or adding more data augmentation.
* **Low Validation Accuracy**: Ensure that your data is properly shuffled and that the labels are correctly formatted.