

# **Project Final Report**

## **Traffic Signs Recognition**

### **GROUP: G**

Jasleen Kaur (C0886477)

Komal Astawala (C0883392)

Mayankkumar Patel (C0883127)

Shreeprada Devaraju (C0887257)

Simran (C0896133)

## **1. Project Objective:**

Utilizing deep learning and Python, this project creates a system for recognizing traffic signs. This project aims to categorize the many traffic signals that are included in an image. This will be accomplished by utilizing Keras to create an accurate deep neural network model for classifying traffic signs. A public dataset with over 50,000 photos of various traffic signs divided into 43 types will be used to train and test the model. The dataset varies, with fewer images in some classes and more images in others. The dataset is approximately 300 MB in size.

## **2. Introduction to Dataset**

### **German Traffic Sign Recognition Benchmark (GTSRB) Dataset:**

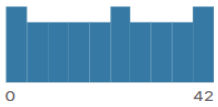


This Kaggle dataset was used by us. 43 distinct classes comprise almost 50,000 photos of various traffic sign types in the dataset utilized for this study. The dataset varies, with fewer images in some classes and more images in others. The collection contains images in four colors (0-red, 1-blue, 2-yellow, 3-white) and five sizes (0-triangle, 1-circle, 2-diamond, 3-hexagon, 4-inverse triangle).

Detail Compact Column

5 of 5 columns

## About this file

This file provides classes meta information about classes provided by this dataset

▲ Path Path to image	∞ ClassId Image class ID	∞ ShapeId Shape of sign (0-red, 1-blue, 2-yellow, 3-white)	∞ ColorId Color of sign (0-triangle, 1-circle, 2-diamond, 3-hexagon, 4-inverse triangle)	▲ SignId Sign ID (by Ukrainian Traffic Rule)
<b>43</b> unique values				3.29 19% 3.3 5% Other (33) 77%
Meta/27.png	27	0	0	1.32
Meta/0.png	0	1	0	3.29
Meta/1.png	1	1	0	3.29

## Python Packages Used:

1. **cv2 (OpenCV):** OpenCV is used to load, manipulate, and analyze images as part of image processing activities. Due to its large library of algorithms for computer vision, it is essential for traffic sign identification tasks like object detection, feature extraction, and image preparation.
2. **NumPy:** NumPy is utilized in array operations and numerical computations. NumPy facilitates the effective management and manipulation of visual data arrays in traffic sign identification, streamlining and optimizing computationally demanding processes including feature extraction, data pretreatment, and model training.
3. **OS:** The operating system module offers features for communicating with it. It is used for file handling, directory navigation, path management to access datasets, store trained models, and arrange processed photos in traffic sign recognition.
4. **Random:** Functions for producing random numbers are available in the random module. Randomness can be used in traffic sign identification tasks like data augmentation (e.g., randomly scaling, or rotating photos) to diversify the training set and strengthen the system's resilience.

## 3. Data Preprocessing

## Image Augmentation Techniques used:

1. **Rotate Image:** To enhance the training data variety and the model's capacity to identify signs from various viewpoints, rotate the input image by a predetermined angle around its center.

2. **Flip Image:** To simulate various traffic sign orientations, flip the supplied image either vertically or horizontally. This augmentation method adds more diversity to the dataset, which strengthens the model's resilience.
3. **Translate Image:** To simulate changes in the locations of the traffic signs inside the frame, move the image both vertically and horizontally. This addition broadens the dataset and enhances the model's capacity to identify indicators at various points within the picture.
4. **Adjust Brightness:** Simulating changes in lighting conditions by adjusting the brightness of the image by scaling the pixel values. By using this method, the model is guaranteed to be able to identify signs in a variety of lighting conditions.
5. **Shear Image:** To imitate perspective shifts, apply a shear transformation to the image, which will distort it along one axis. By adding variances to the dataset, this augmentation strategy strengthens the model's resistance to various viewing angles.
6. **Grayscale Image:** Reduce the amount of color channels in the image by converting it to grayscale. Grayscale images make it easier for the model to concentrate on pertinent information during training by streamlining the input data while maintaining important features.

## 4. Model Implementation

For model implementation, 3 models were tried.

### 1. VGG16

#### ➤ **Model Architecture:**

- Utilizes transfer learning with pre-trained VGG16 CNN, excluding fully connected layers.
- Additional layers for feature extraction and classification: flattening, dense (256 units, ReLU), dropout (rate=0.5), dense (43 units, softmax).

#### ➤ **Compilation:**

- Configured with categorical cross-entropy loss and Adam optimizer.
- Accuracy metric for evaluating training and validation performance.

#### ➤ **Benefits:**

- Transfer learning expedites training and enhances feature extraction.
- Freezing VGG16 layers conserves resources and reduces overfitting risk.
- Dropout regularization aids in generalization for improved performance.

### 2. Logistic Regression:

#### ➤ **Model Architecture:**

- Utilizes logistic regression, a linear classifier for binary classification tasks.
- Input features are scaled using StandardScaler to ensure all features contribute equally to the model.

- Logistic regression model is initialized and trained using max\_iter=1000 for optimization.

➤ **Compilation:**

- No compilation step is required for logistic regression as it is a simple linear model.
- Training is straightforward with the fit method.

➤ **Benefits:**

- Logistic regression is a simple and interpretable model, making it easy to understand the impact of each feature on the prediction.
- Feature scaling improves model performance by ensuring features are on a similar scale.
- The max\_iter parameter allows for fine-tuning the optimization process for better convergence.
- Overall, logistic regression is efficient and suitable for binary classification tasks, especially when the number of features is not extremely large.

### 3. Random Forest Classifier:

- **Model Architecture:**

- Utilizes a Random Forest classifier, an ensemble method combining multiple decision trees for classification.
- Hyperparameters n\_estimators and max\_depth are tuned using GridSearchCV to find the best combination for the model.

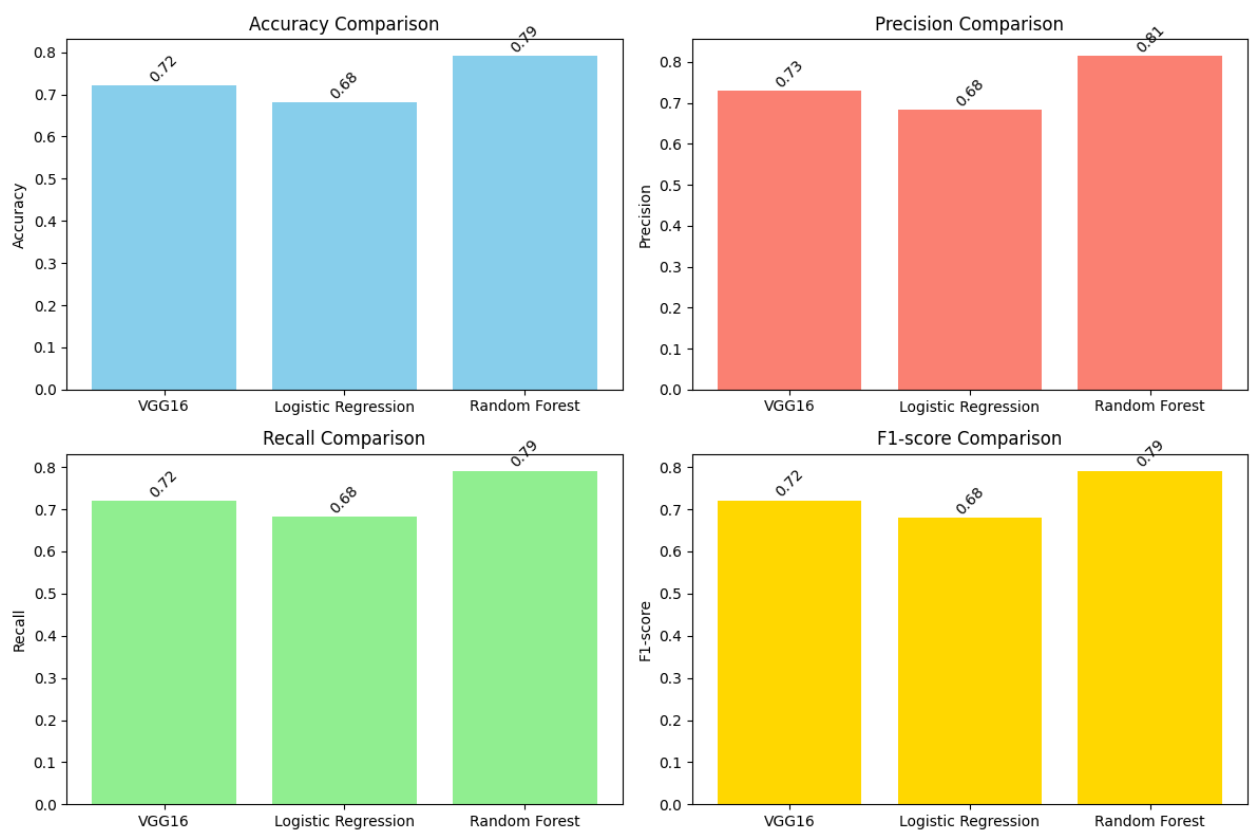
- **Compilation:**

- Random Forest does not require explicit compilation like neural networks.
- Hyperparameters are set and the model is trained using the fit method.

- **Benefits:**

- Random Forest is robust against overfitting due to the ensemble nature of combining multiple decision trees.
- GridSearchCV helps in finding the best hyperparameters, optimizing the model's performance.
- The n\_jobs=-1 parameter enables parallelism, speeding up the grid search process.
- Random Forest is suitable for handling high-dimensional data, such as flattened images, making it a good choice for image classification tasks.

Comparison of VGG16, Logistic Regression and Random Forest Classifier for Traffic Sign Recognition:

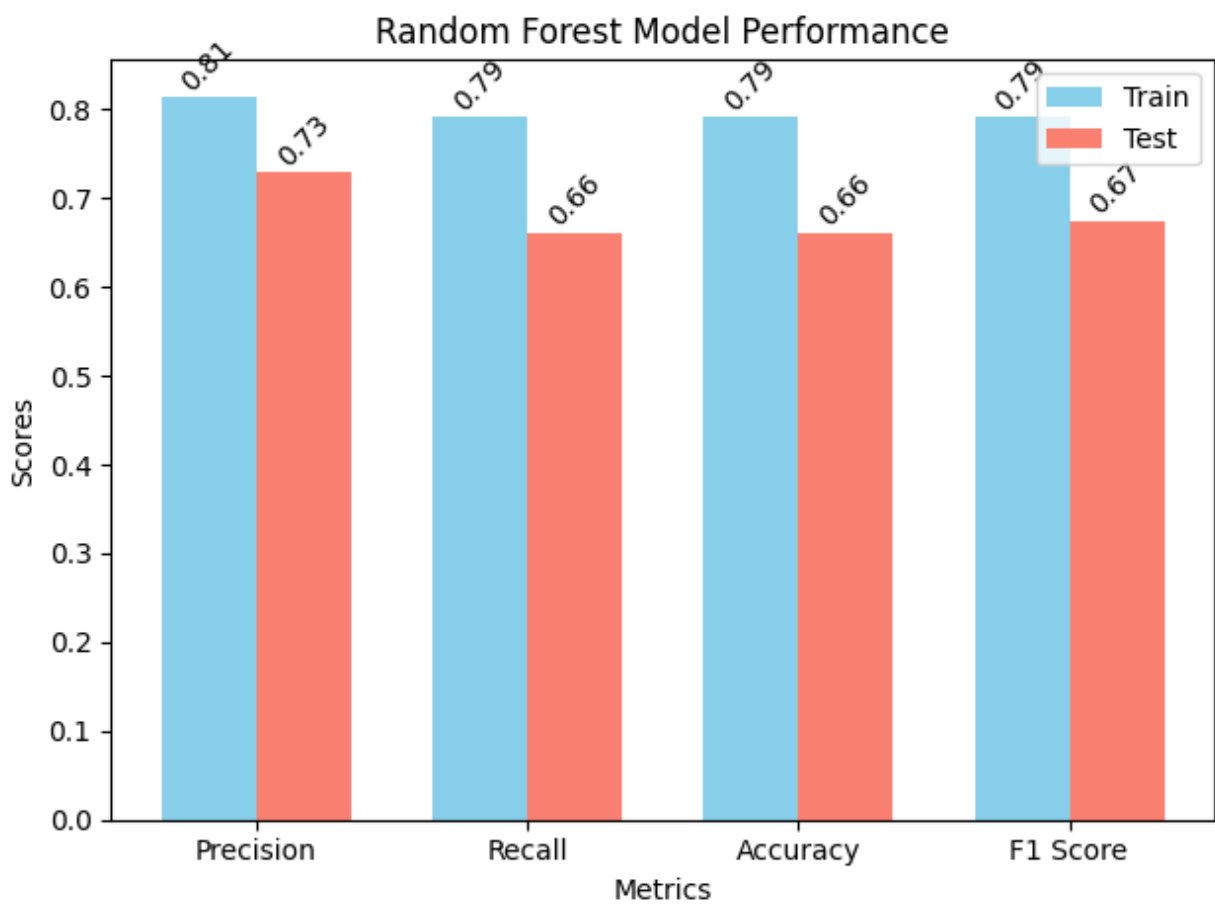


Model	Accuracy	Precision	Recall	F1-Score
VGG16	0.72	0.73	0.72	0.72
Logistic Regression	0.68	0.68	0.68	0.68
Random Forest	0.79	0.81	0.79	0.79

Reason for selecting Random Forest Classifier:

- Random Forest has the highest F1-score (0.79) among the models, indicating a good balance between precision and recall.
- Random Forest has a higher precision (0.81) compared to VGG16 (0.73), which means it makes fewer false positive errors.

5. Metrics Comparison of Train and Test performances for Random Forest Classifier:



Metric	Train	Test
Accuracy	0.79	0.66
Precision	0.81	0.73
Recall	0.79	0.66
F1-Score	0.79	0.67

- The model's performance on the test set is still relatively good, which suggests that the model has learned some meaningful patterns from the data.
- The model's precision is higher than its recall, which means that the model is better at avoiding false positive errors (predicting a positive class when it is actually negative) than

it is at avoiding false negative errors (predicting a negative class when it is actually positive). This may be appropriate depending on the specific task at hand.

## 6. Model Tuning Results:

In our project, we conducted tuning experiments on our random forest classifier:

- **n\_estimators=100:**  
The best model was found to perform optimally with 100 decision trees in the forest. This means that the model combines the predictions of 100 individual decision trees to make its final prediction, which helps in reducing overfitting and improving generalization.
- **max\_depth=None:**  
The best model was found to perform optimally without limiting the maximum depth of the individual decision trees. This allows the trees to grow as deep as necessary to capture the complexity of the data, which can be beneficial for improving the model's performance on the given task.

## 7. User Interface:

### Utilizing .h5 Format for Storing Trained Models:

Storing trained models in the .h5 format offers advantages in terms of versatility, efficiency, and portability. The .h5 format supports various data types, provides efficient storage and compression, and ensures compatibility across different platforms and frameworks. This makes it a convenient choice for managing large models and facilitating easy sharing and deployment.

### Enhancing User Interaction with a Graphical User Interface (GUI) for Traffic Sign Recognition:

#### Introduction to Tkinter:

Tkinter is a standard Python library used for creating graphical user interfaces (GUIs). It offers a simple and beginner-friendly framework for developing desktop applications with interactive elements like buttons, labels, and entry fields.

#### Benefits of Using Tkinter:

- **Ease of Use:** Tkinter is beginner-friendly and easy to learn, making it accessible to developers of all skill levels.

- **Cross-Platform Compatibility:** Tkinter applications can run on various operating systems without modification, ensuring broad compatibility.
- **Integration with Python:** Tkinter seamlessly integrates with Python, allowing developers to leverage Python's rich ecosystem of libraries and tools.
- **Built-in Widgets:** Tkinter provides a wide range of built-in widgets for creating interactive GUI elements.
- **Customization:** Tkinter allows for extensive customization of GUI elements, including colors, fonts, and styles, enabling developers to create visually appealing interfaces.

### Features of the GUI Application:

- **Window Configuration:** The application window is configured with a specific size, title, and background color.
- **Image Upload:** Users can upload an image using the "Upload an image" button, which opens a file dialog for selecting an image file.
- **Image Display:** The uploaded image is displayed in the GUI using a Label widget.
- **Classification:** Users can classify the uploaded image by clicking the "Classify Image" button, triggering a classification process based on a pre-trained model.
- **Result Display:** The predicted class label for the uploaded image is displayed below the image.
- **GUI Layout:** Widgets are arranged using various Tkinter layout managers to achieve the desired layout and positioning.

### Potential Improvements:

- **Enhanced User Feedback:** Provide additional visual feedback to the user during the classification process.
- **Error Handling:** Implement robust error handling to gracefully handle exceptions or unexpected user inputs.
- **Improved Layout and Design:** Enhance the visual appearance of the GUI by refining the layout and applying consistent styling.
- **Interactive Elements:** Incorporate interactive elements to enhance user interaction and functionality.
- **Performance Optimization:** Optimize the classification process for speed and efficiency.
- **Localization and Internationalization:** Support multiple languages and locales to cater to a global audience.

By incorporating these improvements, the GUI application can become more user-friendly, visually appealing, and functional, providing a better overall user experience. Leveraging Tkinter's flexibility and ease of use allows for quick iteration and experimentation to refine the GUI further.



## 8. Test Cases for Traffic Sign Recognition GUI:



### Test Case 1: Image Upload Functionality:

- **Test Case ID:** TSR\_GUI\_TC1
- **Description:** Verify that the user can upload an image using the "Upload" button.
- **Test Steps:**
  - Click on the "Upload" button.
  - Select an image file from the file dialog.
  - Click on the "Open" button in the file dialog.
- **Expected Result:** The selected image should be uploaded and displayed on the GUI.

### Test Case 2: Image Classification:

- **Test Case ID:** TSR\_GUI\_TC2
- **Description:** Verify that the user can classify the uploaded image using the "Classify Image" button.
- **Preconditions:** An image has been successfully uploaded to the GUI.
- **Test Steps:**
  - Click on the "Classify Image" button.
- **Expected Result:** The GUI should display the predicted class label for the uploaded image.

### Test Case 3: Error Handling:

- **Test Case ID:** TSR\_GUI\_TC3
- **Description:** Verify that the GUI handles errors gracefully when the user tries to classify an image without uploading one.
- **Test Steps:**
  - Ensure that no image is uploaded to the GUI.

- **Expected Result:** The GUI should not display classify image button without any uploaded image in the GUI.

#### **Test Case 4: Model Prediction Display:**

- **Test Case ID:** TSR\_GUI\_TC4
- **Description:** Verify that the predicted class label is displayed correctly after classification.
- **Preconditions:** An image has been successfully uploaded and classified.
- **Test Steps:**
  - Check the displayed predicted class label.
- **Expected Result:** The predicted class label should match the expected result based on the uploaded image.

#### **Test Case 5: GUI Responsiveness:**

- **Test Case ID:** TSR\_GUI\_TC5
- **Description:** Verify that the GUI remains responsive during the image upload and classification process.
- **Test Steps:**
  - Upload a large image file.
  - Click on the "Classify Image" button.
- **Expected Result:** The GUI should remain responsive and not freeze or become unresponsive during the upload and classification process.

#### **Test Case 6: Performance Testing:**

- **Test Case ID:** TSR\_GUI\_TC8
- **Description:** Verify that the GUI performance is acceptable under different conditions.
- **Test Steps:**
  - Upload images of varying sizes and complexities.
  - Measure the time taken for image upload and classification.
- **Expected Result:** The GUI should perform efficiently, with minimal lag or delay in image processing.

#### **Test Case 7: Usability Testing:**

- **Test Case ID:** TSR\_GUI\_TC9
- **Description:** Evaluate the usability of the GUI from a user's perspective.
- **Test Steps:**
  - Ask users to perform common tasks (uploading an image, classifying an image) using the GUI.
  - Gather feedback on the user experience.
- **Expected Result:** The GUI should be intuitive and easy to use, with users able to complete tasks without confusion or difficulty.

These test cases cover various aspects of the GUI for the Traffic Sign Recognition project, including functionality, error handling, performance, and usability. Additional test cases can be created to further test the GUI's behavior under different scenarios and edge cases.