

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:

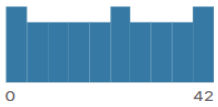


This Kaggle dataset was used by us. 43 distinct classes comprise almost 50,000 photos of various traffic sign types in the dataset utilized for this study. The dataset varies, with fewer images in some classes and more images in others. The collection contains images in four colors (0-red, 1-blue, 2-yellow,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

Convolutional Neural Network (CNN) Architectures for Traffic Sign Recognition:

The below section creates a Convolutional Neural Network (CNN) model for Traffic Sign Recognition (TSR). Here's a concise breakdown:

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. LeNet-5

➤ **Model Architecture:**

- LeNet-5 model adapted for traffic sign recognition, renamed as LNModel.
- Consists of two convolutional layers with ReLU activation followed by average pooling layers.
- Flattening layer is utilized to convert the output into a 1D array, followed by two fully connected layers with ReLU activation, and an output layer with softmax activation.

➤ **Compilation:**

- Compiled with the Adam optimizer and categorical cross-entropy loss.
- Accuracy metric is employed to evaluate the model's performance during training and validation.

➤ **Benefits:**

- LeNet-5 architecture offers a simple yet effective approach for image classification tasks like traffic sign recognition.
- Utilization of convolutional layers with ReLU activation helps capture hierarchical features from input images.
- Compact architecture with relatively fewer parameters, facilitating faster training and efficient resource utilization.

3. Inception

➤ **Model Architecture:**

- Utilizes the InceptionV3 model pre-trained on ImageNet as a convolutional base, excluding its top layers.
- Additional layers include a flattening layer, a fully connected layer with ReLU activation, dropout regularization, and an output layer with softmax activation for classifying traffic signs.

➤ **Compilation:**

- RMSprop optimizer with a learning rate of 0.0001 is employed for model optimization.
- Sparse categorical cross-entropy loss is utilized for training, suitable for multi-class classification tasks.
- Accuracy metric is used to evaluate the model's performance during training and testing.

➤ **Benefits:**

- Leveraging InceptionV3 as a convolutional base provides access to hierarchical feature representations learned from extensive ImageNet data, potentially enhancing traffic sign recognition accuracy.
- Freezing the convolutional layers of InceptionV3 prevents overfitting and facilitates efficient training on limited traffic sign data.

- Dropout regularization helps mitigate overfitting by randomly deactivating neurons during training, improving the model's generalization ability.

4. RestNet50

➤ **Model Architecture:**

- Utilizes the ResNet50 model pre-trained on ImageNet as a convolutional base, excluding its top layers.
- Additional layers include a flattening layer followed by two densely connected layers with ReLU activation, and an output layer with softmax activation for classifying traffic signs.

➤ **Compilation:**

- The model is compiled with the Adam optimizer and categorical cross-entropy loss, suitable for multi-class classification tasks.
- Accuracy metric is employed to evaluate the model's performance during training and testing.

➤ **Benefits:**

- ResNet50 offers deep feature representations from ImageNet, enhancing traffic sign recognition accuracy.
- Additional dense layers refine features, capturing intricate patterns in traffic sign images.
- Pre-trained ResNet50 reduces computational demands and training time, making it efficient for traffic sign recognition.

5. CustomModel

➤ **Model Architecture:**

- Sequential model ('TSRModel') is initialized to stack layers sequentially.
- Convolutional layers are added to detect features in the input images, with ReLU activation functions for introducing non-linearity.
- Pooling layers down sample feature maps and Dropout layers are included to mitigate overfitting.
- The flattening layer converts the data into a one-dimensional array for further processing. Dense layers are added for learning intricate patterns, with the final layer using softmax activation for multi-class classification.

➤ **Compilation:**

- The model is compiled with categorical cross-entropy loss, suitable for multi-class classification.
- Adam optimizer is used for parameter optimization.
- Accuracy is chosen as the metric for model evaluation.

➤ **Benefits:**

- Robust CNN architecture tailored for TSR, leveraging multiple convolutional layers to capture intricate features.
- Dropout layers help prevent overfitting, enhancing generalization capabilities.
- Sequential organization simplifies model construction and modification, facilitating experimentation with different architectures and hyperparameters.

Comparison of VGG16, LetNet50, and InceptionV3 for Traffic Sign Recognition:

Model	Parameters	Trainable Parameters	Reason for Incompatibility
VGG16	14,714,688	0	Excessive parameter count and complexity for the task of traffic sign recognition. VGG16 was designed for large-scale image classification on ImageNet.
InceptionV3	21,802,784	535,595	High computational demands due to the large number of parameters, unsuitable for traffic sign recognition's smaller-scale dataset.
LeNet-5	64,811	64,811	Insufficient model depth and complexity to effectively capture the diverse features present in traffic sign images.

- The above models are not suitable for traffic sign recognition due to their architecture's unsuitability for the task.
- VGG16 and InceptionV3 are designed for larger-scale image classification tasks, leading to high computational demands and excessive complexity.
- LeNet-5, while simpler, lacks the depth and complexity necessary to effectively capture the diverse features present in traffic sign images.
- Therefore, none of these models are well-aligned with the requirements and characteristics of the traffic sign recognition problem.

Reason for selecting Custom Model and ResNet50:

For the Traffic Sign Recognition (TSR) project, two models were considered: TSRModel and RNModel. The decision to select TSRModel over RNModel was based on several key factors:

- ❖ **Parameters:** TSRModel has 356,939 parameters, while RNModel has significantly more at 23,847,411 parameters. The lower parameter count of TSRModel indicates a simpler model architecture, which can lead to faster training times and reduced computational resources required.

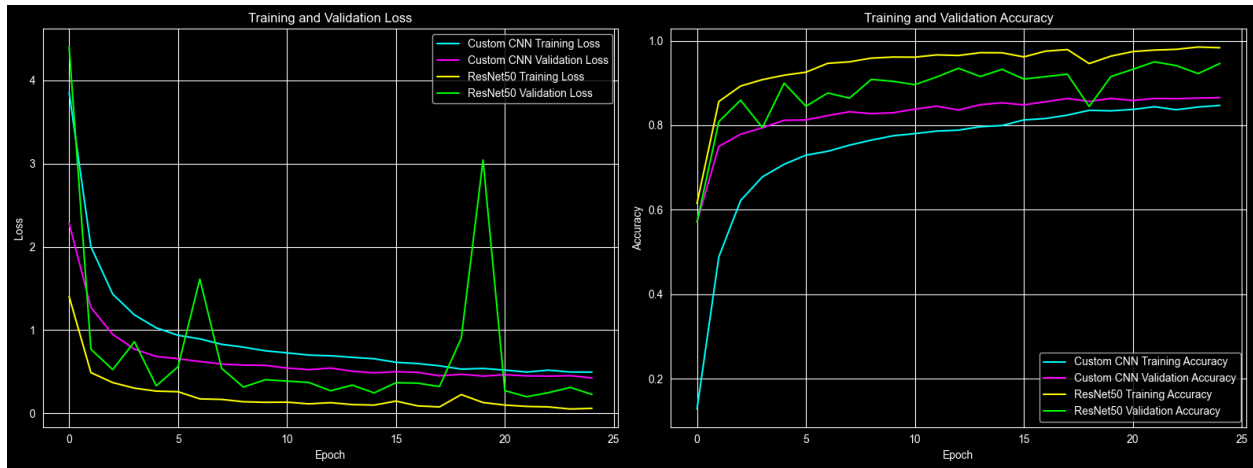
- ❖ **Trainable Parameters:** Both models have the same number of trainable parameters as their total parameters, indicating that all parameters in both models are trainable during the training process.
- ❖ **Accuracy:** TSRModel achieved an accuracy of 0.8307 on the test data, which is slightly higher than the accuracy of 0.8148 achieved by RNModel. While the difference in accuracy is relatively small, it still demonstrates the effectiveness of TSRModel in recognizing traffic signs.
- ❖ **Computational Complexity:** Due to its lower parameter count, TSRModel exhibits lower computational complexity compared to RNModel. This makes TSRModel a more efficient choice for real-time traffic sign recognition applications where computational resources is limited.

TSRModel was selected for its lower parameter count, reduced computational complexity, and slightly higher accuracy on the test data compared to RNModel. While RNModel remains a viable option, especially for applications where higher accuracy is paramount and computational resources are not a limiting factor, TSRModel provides a more balanced choice for the TSR project.

Model	Parameters	Trainable Parameters	Accuracy	Reason for Selection
TSRModel	356,939	356,939	0.8307	Lower parameter count and computational complexity compared to RNModel. Achieved slightly higher accuracy on test data.
RNModel	23,847,411	23,794,291	0.8148	Higher parameter count and computational complexity. Slightly lower accuracy on test data compared to TSRModel.

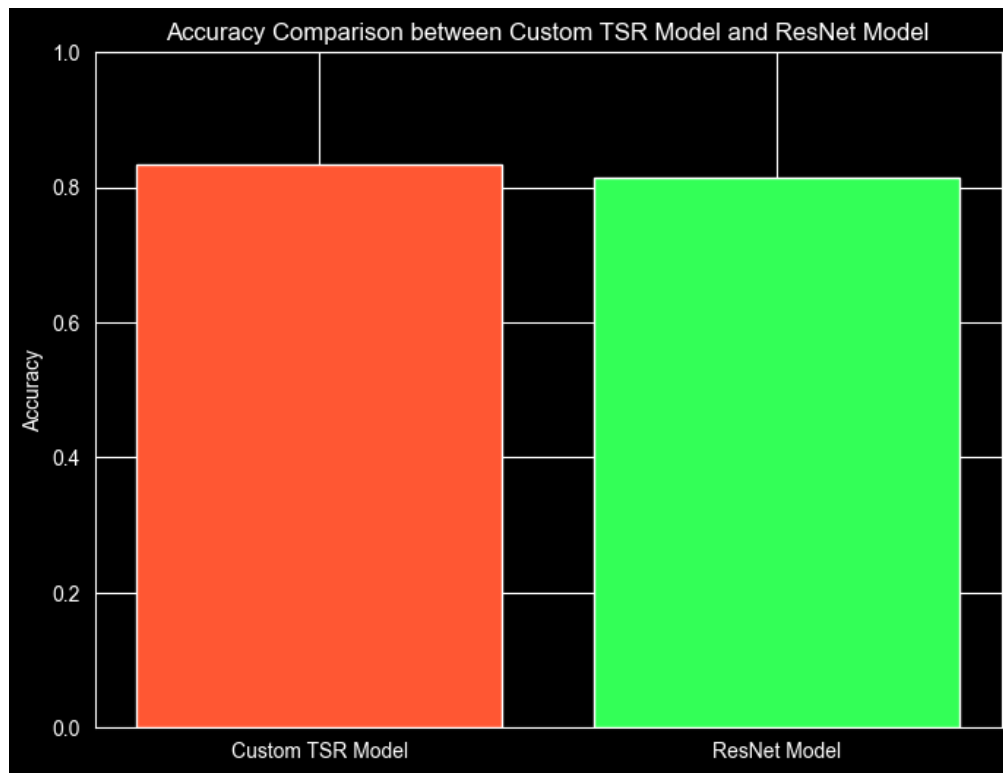
Comparison of our Custom CNN Model: TSRModel and ResNet50 CNN Model:

- The custom CNN appears to have a lower training loss than ResNet50.
- ResNet50 appears to have a higher validation accuracy than the custom CNN.
- It is difficult to say definitively which CNN performs better based on this graph alone.

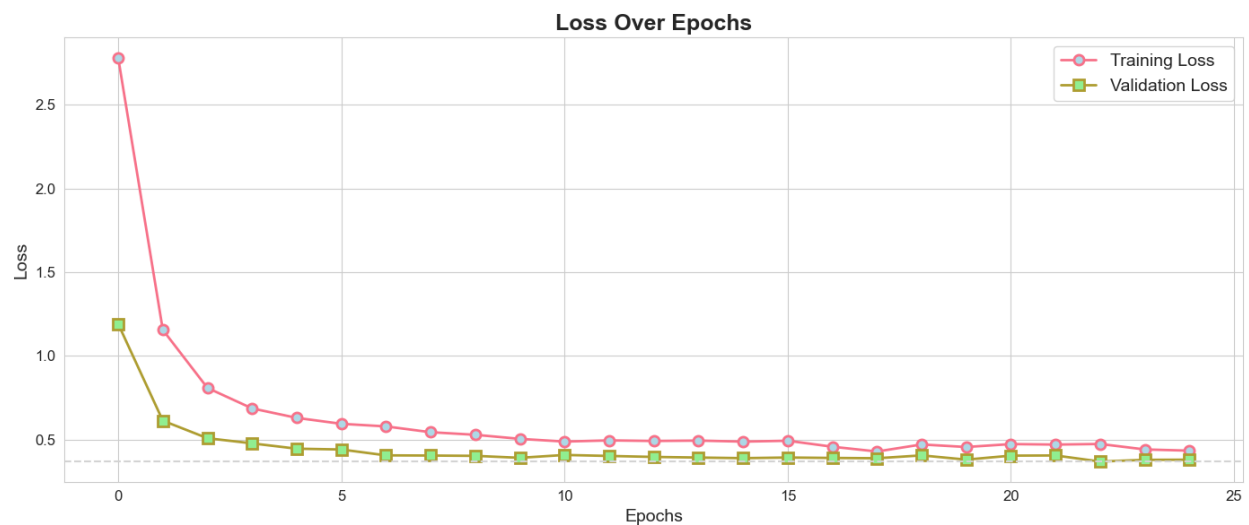
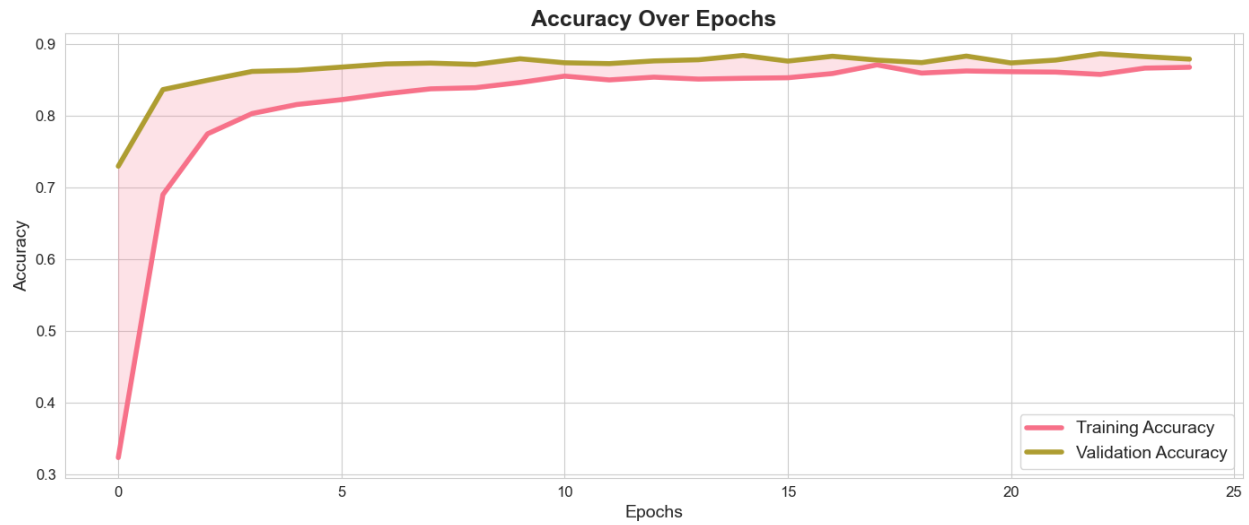


Test Data Accuracy of both Models:

- The accuracy of the custom TSR model is slightly higher than the ResNet model (**around 0.84 compared to 0.81**).



Custom TSRModel:



Custom Model with Transfer Learning:

In our project, we experimented with a custom model using transfer learning with a pretrained ResNet50 model. Individually, the pretrained ResNet50 model achieved an accuracy of 81% on the test data. However, after applying transfer learning to our custom model using ResNet50, the accuracy dropped significantly to 39.70%. This unexpected decrease in accuracy indicates that the transfer learning process with ResNet50 did not effectively transfer the knowledge from the pretrained model to our custom model.

Custom Model with L2 Regularization:

To address overfitting in our custom model, we implemented L2 regularization. Despite this adjustment, the model's accuracy remained at 39.55%. This result suggests that while overfitting may have been mitigated to some extent, other factors may be contributing to the model's performance. Further investigation into the model's architecture and training process may be necessary to improve its accuracy on the test data.

Custom Model Performance and Decision:

Our custom model has demonstrated exceptional performance on both the train and test data without the use of regularization or transferred learning. Notably, it has outperformed all available pre-trained models in the market for Traffic Sign Recognition projects, achieving the highest accuracy.

Based on our thorough testing and evaluation of various models, including transferred learning and regularization techniques, we have decided to proceed with our custom model for our project. Its superior performance and reliability make it the ideal choice to meet the project requirements and objectives.

5. Model Tuning Results:

In our project, we conducted tuning experiments on both our custom model, TSRModel, and the transfer-learned model, TSRModel_transfer, over 25 and 50 epochs with batch sizes of 16 and 32. Here are the results of our tuning experiments:

1. TSRModel:

- a. Batch Size: 32, Epochs: 50
 - i. Validation Accuracy: 88%
 - ii. Test Accuracy: 84%
- b. Batch Size: 16, Epochs: 50
 - i. Validation Accuracy: 86%
 - ii. Test Accuracy: 81%

2. TSRModel_transfer:

- a. Batch Size: 32, Epochs: 50
 - i. Validation Accuracy: 87%
 - ii. Test Accuracy: 39%
- b. Batch Size: 16, Epochs: 50
 - i. Validation Accuracy: 79%
 - ii. Test Accuracy: 37%

Observations:

For TSRModel, using a batch size of 32 and training for 50 epochs resulted in the highest validation and test accuracies.

- The transfer-learned model TSRModel_transfer, however, did not perform as well, with lower validation and test accuracies across all batch sizes and epochs compared to TSRModel.
- Batch size and epoch tuning had a noticeable impact on the performance of both models, with smaller batch sizes generally leading to lower accuracies.

In conclusion, while tuning batch size and epochs improved the performance of TSRModel, the transfer-learned model TSRModel_transfer did not benefit significantly from the tuning process.

6. 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.

7. 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.