# Traffic Sign Recognition

## DEEP LEARNING FINAL PROJECT


Group 8:
Aditya Koshti - 101504567
Ayushi Patel - 101450798
Mahesh Dadheech - 101501168
Sakshi Sareen - 101471328
Shivani - 101504534

# OVERVIEW


- Problem Statement
- Dataset Overview
- Data Preprocessing
- Model 1 - CNN
- Model 2 - MobileNetV2
- Model 3 - ResNet
- Hyperparameter Tuning
- Challenges and Solutions
- GUI Demonstration
- Conclusion

# PROBLEM STATEMENT


**Challenges Addressed:**

Traffic signs provide critical information to drivers, such as speed limits, warnings, and directional guidance. An automatic traffic sign recognition system can help in reducing human errors, enhancing navigation systems, and improving the overall safety of driving. With the advent of deep learning techniques,  it has shown great promise in image recognition tasks, making them ideal for traffic sign recognition.

**Scope:**

- Focus on standard traffic signs.
- Train the model on a diverse dataset to ensure it can handle real-world variability in sign appearance.

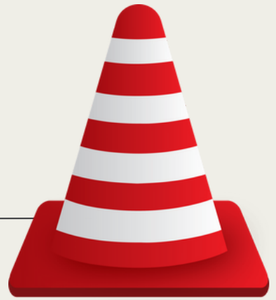
# DATASET OVERVIEW

Traffic Sign Recognition Benchmark (TSRB) is a comprehensive dataset designed for the multi-class, single-image classification of traffic signs. This dataset was introduced as part of a competition held at the International Joint Conference on Neural Networks (IJCNN) in 2011. Some of the key properties of the dataset:

- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total
- Large real-life dataset

Dataset link: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?select=Meta

# DATA PREPROCESSING



- We have converted the list into numpy arrays for feeding to the model.

- The shape of data is (39209, 30, 30, 3) meaning that there are 39,209 images of size 30×30 pixels and the last 3 means the data contains coloured images (RGB value).

- With the sklearn package, we use the **train_test_split()** method to split training and testing data.

- From the keras.utils package, we use **to_categorical** method to convert the labels present in **y_train and t_test** into one-hot encoding.

# MODEL 1 : CONVOLUTIONAL NEURAL NETWORK(CNN)

## Model Architecture

**Layer 1-2:** Convolutional Layers (32 filters, 5x5 kernel, ReLU activation)
**Layer 3:** MaxPooling (2x2 pool size)
**Layer 4:** Dropout (rate: 0.25)
**Layer 5-6:** Convolutional Layers (64 filters, 3x3 kernel, ReLU activation)
**Layer 7:** MaxPooling (2x2 pool size)
**Layer 8:** Dropout (rate: 0.25)
**Layer 9:** Flatten
**Layer 10:** Dense (256 units, ReLU activation)
**Layer 11:** Dropout (rate: 0.5)
**Layer 12:** Dense (43 units, Softmax activation)

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 46, 46, 32) | 2,432 |
| conv2d_1 (Conv2D) | (None, 42, 42, 32) | 25,632 |
| max_pooling2d (MaxPooling2D) | (None, 21, 21, 32) | 0 |
| dropout (Dropout) | (None, 21, 21, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 19, 19, 64) | 18,496 |
| conv2d_3 (Conv2D) | (None, 17, 17, 64) | 36,928 |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| dropout_1 (Dropout) | (None, 8, 8, 64) | 0 |
| flatten (Flatten) | (None, 4096) | 0 |
| dense (Dense) | (None, 256) | 1,048,832 |
| dropout_2 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 43) | 11,051 |

## Model Compilation

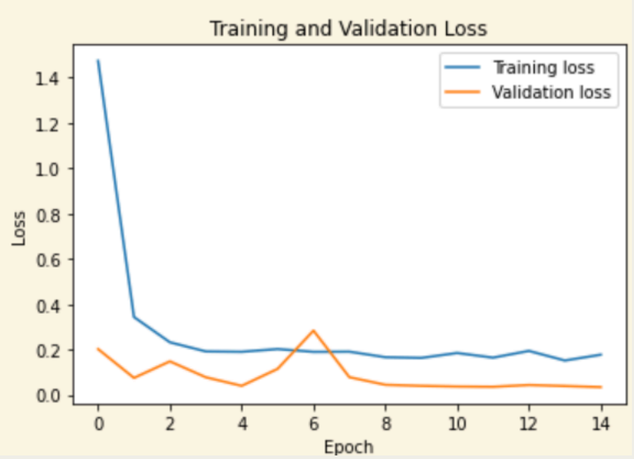
**Optimizer:** ALoss Function: Categorical Crossentropy
**Metrics:** Accuracy

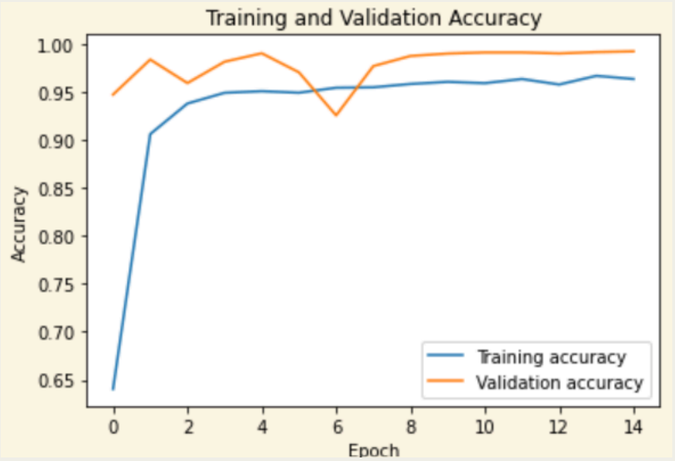
## Training and Validation

**Epochs**: 15
**Batch Size:** 32
**Validation Data:** 20% of the dataset

## Results

**Accuracy : 94.41%**



# CNN - HYPERPARAMETER TUNING WITH OPTUNA

Optuna is an open-source hyperparameter optimization framework in Python. It provides a simple yet powerful API for tuning the hyperparameters of machine learning models. Optuna uses various optimization algorithms to efficiently search for the best set of hyperparameters, improving the performance of machine learning models.

For 40 trials:
Best accuracy with params
conv_filters_1
conv_filters_2
conv_filters_3
dense_units
dropout_rate
learning_rate

**Accuracy after fine tuning: 98.75%**

```
    image = image.resize((30, 30))
    data.append(np.array(image))

X_test = np.array(data)
pred = np.argmax(model.predict(X_test), axis=1)

# Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))

model.save("traffic_classifier_best.h5")
```

```
[I 2024-05-31 13:19:22,426] Trial 33 finished with value: 0.9945167303085327 and parameters: {'conv_filters_1': 45, 'conv_filters_2': 37, 'conv_filt
s_3': 128, 'dense_units': 373, 'dropout_rate': 0.31245489708221424, 'learning_rate': 0.0001623910868902432}. Best is trial 27 with value: 0.996301945
242004.
[I 2024-05-31 13:29:29,743] Trial 34 finished with value: 0.993751585483551 and parameters: {'conv_filters_1': 52, 'conv_filters_2': 50, 'conv_filte
_3': 105, 'dense_units': 294, 'dropout_rate': 0.36275463478438275, 'learning_rate': 7.53764795368845e-05}. Best is trial 27 with value: 0.996301949
2004.
[I 2024-05-31 13:38:08,009] Trial 35 finished with value: 0.993241548538208 and parameters: {'conv_filters_1': 39, 'conv_filters_2': 48, 'conv_filte
_3': 122, 'dense_units': 388, 'dropout_rate': 0.22575831664787566, 'learning_rate': 0.0004539875652405387 7}. Best is trial 27 with value: 0.996301945
242004.
[I 2024-05-31 13:48:03,621] Trial 36 finished with value: 0.9928589463233948 and parameters: {'conv_filters_1': 44, 'conv_filters_2': 56, 'conv_filt
s_3': 114, 'dense_units': 437, 'dropout_rate': 0.34210445909012, 'learning_rate': 0.00022909950243692426}. Best is trial 27 with value: 0.996301949
2004.
[I 2024-05-31 13:57:53,797] Trial 37 finished with value: 0.9880132675170898 and parameters: {'conv_filters_1': 36, 'conv_filters_2': 60, 'conv_filt
s_3': 76, 'dense_units': 464, 'dropout_rate': 0.26030107544346914, 'learning_rate': 0.0003675894514572278}. Best is trial 27 with value: 0.996301949
42004.
[I 2024-05-31 14:06:30,863] Trial 38 finished with value: 0.987503170967102 and parameters: {'conv_filters_1': 34, 'conv_filters_2': 47, 'conv_filte
_3': 81, 'dense_units': 331, 'dropout_rate': 0.30410397683517987, 'learning_rate': 0.0009580126606726513}. Best is trial 27 with value: 0.996301949
```

# MODEL 2 : MOBILENETV2(PRE-TRAINED)

## Model Architecture

**Layer 1:** MobileNetV2 base model without the top layer
**Layer 2:** GlobalAveragePooling2D
**Layer 3:** Dense (512 units, ReLU activation)
**Layer 4:** Dropout (rate: 0.5)
**Layer 5:** Dense (256 units, ReLU activation)
**Layer 6:** Dropout (rate: 0.5)
**Layer 7:** Dense (43 units, Softmax activation)

## Model Compilation

**Optimizer:** Adam
**Loss Function**: Categorical Crossentropy
**Metrics:** Accuracy

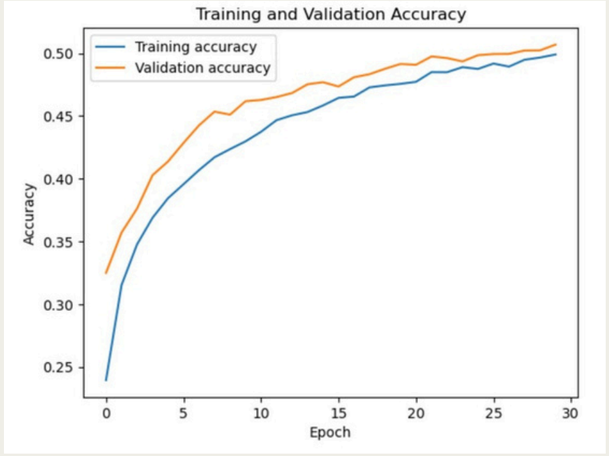
## Training and Validation

**Epochs**: 10
**Batch Size:** 32
**Validation Data:** 20% of the dataset

## Results

**Accuracy : 50 %**

```
981/981 ━━━━━━━━━━━━━━━━━━━━ 31s 32ms/step - accuracy: 0.4467 - loss: 1.7829 - val_accuracy: 0.4652 - val_loss: 1.7407
Epoch 13/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 32s 33ms/step - accuracy: 0.4532 - loss: 1.7544 - val_accuracy: 0.4684 - val_loss: 1.7310
Epoch 14/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 34ms/step - accuracy: 0.4596 - loss: 1.7352 - val_accuracy: 0.4754 - val_loss: 1.7137
Epoch 15/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 34ms/step - accuracy: 0.4606 - loss: 1.7229 - val_accuracy: 0.4770 - val_loss: 1.6973
Epoch 16/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 34ms/step - accuracy: 0.4697 - loss: 1.7134 - val_accuracy: 0.4736 - val_loss: 1.6920
Epoch 17/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 33s 34ms/step - accuracy: 0.4617 - loss: 1.6986 - val_accuracy: 0.4810 - val_loss: 1.6862
Epoch 18/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 35s 36ms/step - accuracy: 0.4770 - loss: 1.6748 - val_accuracy: 0.4834 - val_loss: 1.6768
Epoch 19/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 36s 37ms/step - accuracy: 0.4790 - loss: 1.6662 - val_accuracy: 0.4878 - val_loss: 1.6660
Epoch 20/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 35s 36ms/step - accuracy: 0.4818 - loss: 1.6509 - val_accuracy: 0.4916 - val_loss: 1.6676
Epoch 21/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 33s 34ms/step - accuracy: 0.4790 - loss: 1.6562 - val_accuracy: 0.4909 - val_loss: 1.6606
Epoch 22/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 34ms/step - accuracy: 0.4879 - loss: 1.6299 - val_accuracy: 0.4976 - val_loss: 1.6571
Epoch 23/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 36s 37ms/step - accuracy: 0.4896 - loss: 1.6201 - val_accuracy: 0.4963 - val_loss: 1.6554
Epoch 24/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 33s 34ms/step - accuracy: 0.4908 - loss: 1.6197 - val_accuracy: 0.4936 - val_loss: 1.6499
Epoch 25/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 36s 36ms/step - accuracy: 0.4921 - loss: 1.6085 - val_accuracy: 0.4986 - val_loss: 1.6399
Epoch 26/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 34ms/step - accuracy: 0.4982 - loss: 1.5995 - val_accuracy: 0.4995 - val_loss: 1.6481
Epoch 27/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 35ms/step - accuracy: 0.4920 - loss: 1.5967 - val_accuracy: 0.4996 - val_loss: 1.6393
Epoch 28/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 34s 35ms/step - accuracy: 0.4911 - loss: 1.5991 - val_accuracy: 0.5023 - val_loss: 1.6328
Epoch 29/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 36s 36ms/step - accuracy: 0.5020 - loss: 1.5845 - val_accuracy: 0.5024 - val_loss: 1.6317
Epoch 30/30
981/981 ━━━━━━━━━━━━━━━━━━━━ 35s 36ms/step - accuracy: 0.5066 - loss: 1.5662 - val_accuracy: 0.5070 - val_loss: 1.6263
```



# MOBILENETV2 - HYPERPARAMETER TUNING WITH OPTUNA

The parameters of the best trial included 512 units with dropout.

By employing Optuna for hyperparameter tuning, we systematically searched for the optimal configuration of the Dense layer units and the Dropout rate. This resulted in a significant improvement in model accuracy, achieving 87.26% compared to the 50% accuracy obtained without tuning.

**Results**

**Accuracy : 87.26 %**

```
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with
 the appropriate compiler flags.
[I 2024-05-30 21:31:09,128] Trial 0 finished with value: 0.8108900785446167 and parameters: {'units': 512, 'drop
out': 0.3417805288717809}. Best is trial 0 with value: 0.8108900785446167.
[I 2024-05-30 21:32:27,289] Trial 1 finished with value: 0.8584544658660889 and parameters: {'units': 512, 'drop
out': 0.6702881706565329}. Best is trial 1 with value: 0.8584544658660889.
[I 2024-05-30 21:33:43,469] Trial 2 finished with value: 0.87260901927948 and parameters: {'units': 512, 'dropou
t': 0.6302430378080803}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:35:01,245] Trial 3 finished with value: 0.8459576368331909 and parameters: {'units': 512, 'drop
out': 0.45352502194561145}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:35:44,369] Trial 4 finished with value: 0.8279775381088257 and parameters: {'units': 256, 'drop
out': 0.3063588762480089}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:37:00,975] Trial 5 finished with value: 0.790614664554596 and parameters: {'units': 512, 'dropo
ut': 0.5823997263965413}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:37:44,734] Trial 6 finished with value: 0.8238969445228577 and parameters: {'units': 256, 'drop
out': 0.35926001128598406}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:38:28,160] Trial 7 finished with value: 0.8181586265563965 and parameters: {'units': 256, 'drop
out': 0.6828472014160054}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:39:47,889] Trial 8 finished with value: 0.8282325863838196 and parameters: {'units': 512, 'drop
out': 0.5453619412340747}. Best is trial 2 with value: 0.87260901927948.
[I 2024-05-30 21:41:08,491] Trial 9 finished with value: 0.8353736400604248 and parameters: {'units': 512, 'drop
out': 0.5421486454678297}. Best is trial 2 with value: 0.87260901927948.
Number of finished trials:  10
Best trial:
  Value:  0.87260901927948
  Params:
    units: 512
```

# MODEL 3 - RESNET (PRETRAINED)

**Model Architecture:**

**Layer 1-4:** Convolutional Layers with various filters and kernel sizes, leveraging ResNet-50's deep residual learning framework.

**Layer 5:** Global Average Pooling to reduce dimensionality while maintaining essential features.

**Layer 6:** Dense Layer (256 units, ReLU activation)

**Layer 7:** Dropout (rate: 0.5) to prevent overfitting.

**Layer 8:** Output Dense Layer (43 units, Softmax activation) for classifying 43 types of traffic signs.

**Model Compilation:**

**Optimizer:** Adam

**Loss Function:** Categorical Crossentropy

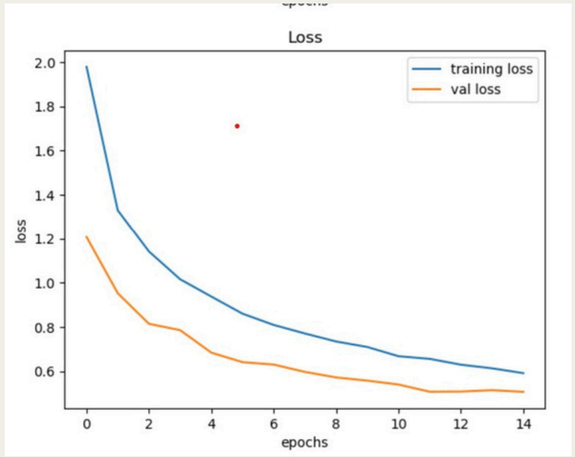
**Metrics**: Accuracy

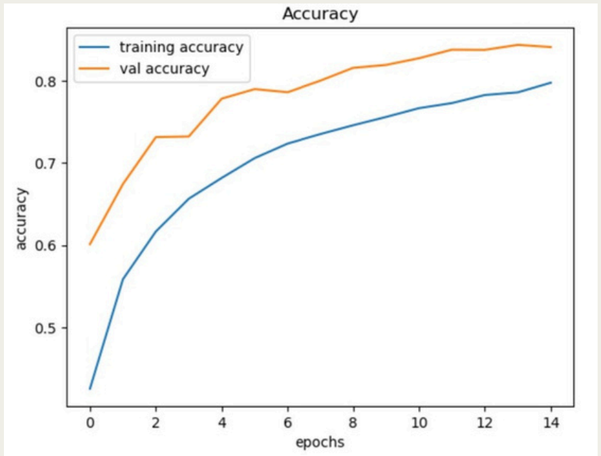
**Training and Validation:**

**Epochs:** 15

**Batch Size:** 32

**Validation Data:** 20% of the dataset




**Results:**

**Accuracy:** 84.15

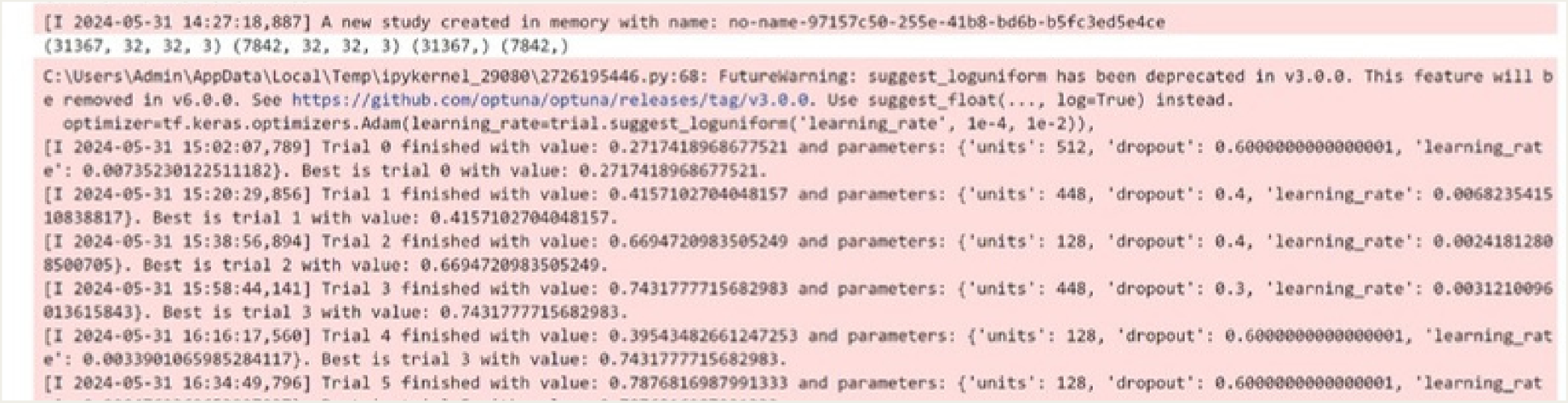
**Observations**: The training loss decreased significantly over epochs, suggesting effective learning, but the validation loss plateau was higher, indicating potential overfitting.

# RESNET- HYPERPARAMETER TUNING WITH OPTUNA

**Key Hyperparameters Tuned:**

- Number of Dense Units: Adjusted the number of neurons in the dense layer (ranging from 128 to 512) to find the optimal size for feature interpretation post-convolution.
- Dropout Rate: Tuned between 0.3 and 0.7 to identify the best rate that prevents overfitting while maintaining sufficient model complexity.
- Learning Rate: Employed a logarithmic scale (from 1e-4 to 1e-2) to fine-tune the learning rate for the Adam optimizer, ensuring efficient and effective model training dynamics.

Result- 78.76%



# CHALLENGES

- **Class Imbalance**: The dataset has an unequal distribution of classes, with some classes having very few samples. This can lead to biased models that perform well on majority classes but poorly on minority classes.
- **Variability in Lighting and Weather**: The dataset contains images taken under different lighting and weather conditions, which can affect the model's performance.
- **Similarity between Classes**: Some traffic signs have similar designs, making it difficult for the model to distinguish between them.
- **Overfitting**: The models may overfit the training data, especially if the models are complex.

# SOLUTIONS

**Class Imbalance:**

- We noticed that the dataset is not highly imbalanced and the model’s performance was decent so we did not use any specific technique to address this problem of class imbalance

**Variability in Lighting and Weather:**

- We resized the images to a fixed size (30x30) or (32x32) depending on the model's requirement, which helped reduce the impact of varying lighting conditions

**Similarity between Classes:**

- The CNN architecture and the use of convolutional and pooling layers can help extract features that distinguish between similar classes so we did not use any specific technique to address the problem of class similarity

**Overfitting:**

- Dropout layers were used to prevent overfitting. Dropout randomly sets a fraction of the output units to zero during training, which helps prevent the model from overfitting.
- We trained the model for a fixed number of epochs instead of using early stopping due to computational limitations.
- Hyperparameter tuning was performed using Optuna, which helped find the best combination of hyperparameters to prevent overfitting.

# TRAFFIC SIGN CLASSIFICATION GUI

Purpose: This GUI allows users to upload images of traffic signs and classify them using a pre-trained neural network model.

- Tools Used:

Tkinter: For creating the GUI.

PIL (Pillow): For handling image operations.

NumPy: For numerical operations on image data.

Keras (TensorFlow): For loading and using the pre-trained model.

- Key Features

Upload Button: Allows users to upload an image of a traffic sign.

Classify Button: Classifies the uploaded image using the pre-trained model.

Result Display: Shows the classified traffic sign label.

# GUI DEMONSTRATION

# FUTURE SCOPE

- This models can further be used by an app, that uses device’s camera to detect the sign.
- We also plan to add a text-to-speech feature in the GUI itself.
- Additionally, we plan to add a feature in the GUI which will allow the user to translate the classified sign into desired language.

# CONCLUSION

In this project, we explored the development of a traffic sign recognition system using various deep learning models. We trained and evaluated multiple models, including MobileNet, ResNet, and our custom CNN architecture, on a dataset of traffic signs. Our goal was to determine the most accurate model for this task.

Model Accuracies

- MobileNet: 87.26%
- ResNet: 84.15%
- Custom CNN: 96.2%

Our results show that the custom CNN architecture outperformed the other models, achieving an accuracy of 96.2%. This demonstrates the effectiveness of our model in recognizing traffic signs.

# Thank you!