Smart Traffic Sign Recognition Using Convolutional Neural Networks

Submitted by

Hridhya Haridas (DA&DS)

1. Summary

This project presents a **Smart Traffic Sign Recognition System** designed to accurately classify road signs using Deep Learning. The solution uses a **Convolutional Neural Network (CNN)** trained on 30 distinct traffic sign categories and is deployed using a **Streamlit web interface** for real-time predictions.

The model demonstrates strong performance, achieving **~86% validation accuracy**, and the deployed application allows users to upload an image and instantly receive predictions with confidence scores. The project builds a solid foundation for future advancements such as **real-time video detection, mobile deployment, and multilingual audio alerts**, making it highly relevant for autonomous vehicles and modern transportation systems.

2. Project Objectives

The project is designed with the following clear objectives:

2.1 Build a Robust Classification Model

Develop a CNN capable of identifying multiple traffic sign classes with high reliability.

2.2 Dataset Structuring & Preprocessing

Clean, organize, augment, and normalize image data to ensure high-quality training.

2.3 Model Evaluation

Assess accuracy, loss curves, confusion matrix, and generalization capability.

2.4 Application Deployment

Create an easy-to-use Streamlit interface for real-time traffic sign recognition.

2.5 Scalability & Future Integration

Design the system such that it can be extended to video streams, mobile apps, and voice-enabled outputs.

3. Technology Stack

Category	Tools/ Frameworks	Purpose
Deep Learning	CNN Architecture	Feature extraction & classification
Frameworks	TensorFlow, Keras	Model training & optimization
Interface	Streamlit	Deployment & real-time prediction
Data Processing	OpenCV, NumPy, Pandas	Image handling, transformations, array operations
Visualization	Matplotlib, Seaborn	Training curves, confusion matrix, error analysis

---

4. Dataset & Preprocessing

4.1 Dataset Characteristics

- **Total Classes:** 30
- **Image Size:** 180×180 pixels
- **Normalization:** Pixel scaling to [0,1]
- **Format:** Folder-wise class-separated images

4.2 Dataset Distribution

Dataset Type	Purpose	Images	Classes
Training	Learn model parameters	504	30
Validation	Tune hyperparameters	149	30
Testing	Evaluate final accuracy	285	30

4.3 Preprocessing Steps

- **Data Augmentation**

- Random horizontal flip
- Random rotation (±10%)
- Random zoom (10%)

```
# Data Augmentation Layer
# ------------------------------
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1)
])
```

- **Label Encoding**
  Converts folder names → numerical class IDs
- **Image Resizing**
  Maintains input shape consistency for CNN

5. CNN Model Architecture

5.1 Feature Extraction Layers

Layer Type	Configuration	Purpose
Rescaling	Normalize to [0,1]	Standardizes input
Conv2D	16 → 32 → 64 filters	Learns edges → textures → shapes
MaxPooling2D	Pool size 2×2	Reduces spatial dimensions

5.2 Classification Layers

Layer	Details	Purpose
Flatten	Convert feature maps to vector	Feed into dense layers
Dropout	0.2	Reduces overfitting
Dense	128 units	Pattern learning

Layer	Details	Purpose
Output Dense	30 units + Softmax	Class probability distribution

```
model = Sequential([
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dropout(0.2),
    layers.Dense(128),
    layers.Dense(len(road_type))

])
```

5.3 Model Training Details

Parameter	Value	Description
Optimizer	Adam	Adaptive learning rate
Loss	Sparse Categorical Crossentropy	Multi-class classification
Metric	Accuracy	Performance measure
Epochs	15	Early stopping applied
Callbacks	EarlyStopping, Checkpoint	Prevent overfitting & save best model

5. Model Compilation and Training

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
epochs_size = 15
history = model.fit(data_train, validation_data=data_validation, epochs=epochs_size)
```

```
Epoch 1/15
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 154ms/step - accuracy: 0.1409 - loss: 3.8288 - val_accuracy:
Epoch 2/15
```

6. Model Performance Summary

- **Training Accuracy:** ~100%
- **Validation Accuracy: 85.9% – 86.6%**
- **Observation:**
  - Slight overfitting is noticed
  - Model performs strongly on unseen data
- **Confusion Matrix:**
  Shows class-wise prediction distribution and misclassification points

```python
# Evaluate model on test set
test_loss, test_accuracy = model.evaluate(data_test)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Get predictions and true labels
y_true = []
y_pred = []
for images, labels in data_test:
    preds = model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(preds, axis=1))

# Classification report
print(classification_report(y_true, y_pred, target_names=road_type))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=road_type, yticklabels=road_type)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

accuracy			0.54	285
macro avg	0.53	0.55	0.49	285
weighted avg	0.56	0.54	0.48	285

7. Prediction Pipeline & Deployment

7.1 Prediction Workflow

1. Upload or capture image
2. Resize to 180×180
3. Convert to NumPy array
4. Apply normalization

5. Run through CNN
6. Softmax applied for probability scores
7. Return predicted class + confidence

```
image = r"C:\Users\Ajay\Downloads\RoadSigns\TEST\16\016_1_0026_1_j.png"

image = tf.keras.utils.load_img(image, target_size=(img_height,img_width))
img_arr = tf.keras.utils.array_to_img(image)
img_bat=tf.expand_dims(img_arr,0)

predict = model.predict(img_bat)

score = tf.nn.softmax(predict)

print('road type in image is {} with accuracy of {:0.2f}'.format(road_type[np.argmax(score)],np.max(score)*100))

plt.imshow(image, cmap='gray')
plt.title('Predicted: {}(Accuracy: {:0.2f}%)'.format(road_type[np.argmax(score)], np.max(score) * 100))
plt.show()
```

7.2 Hyperparameter Tuning (Random Search)

Parameter	Best Value
Filters	32, 96, 192
Kernel Sizes	5, 3
Dropout	0.4
Dense Units	256
Learning Rate	0.001
Epochs	30

Output Accuracy: ~14%

→ Indicates overfitting and insufficient dataset size during tuning run.

→ Further augmentation and tuning needed.

```
Trial 8 Complete [00h 04m 24s]
val_accuracy: 0.05999999865889549

Best val_accuracy So Far: 0.14000000059604645
Total elapsed time: 00h 25m 06s
Best hyperparameters: {'filters_1': 32, 'kernel_size_1': 5, 'filters_2': 96, 'kernel_size_2': 3, 'filters_3': 192
Epoch 1/30
13/13 ━━━━━━━━━━━━━━━━━━━━ 11s 769ms/step - accuracy: 0.0198 - loss: 3.4089 - val_accuracy: 0.0200 - val_loss: 3.
Epoch 2/30
13/13 ━━━━━━━━━━━━━━━━━━━━ 10s 767ms/step - accuracy: 0.0272 - loss: 3.3982 - val_accuracy: 0.0200 - val_loss: 3.
```

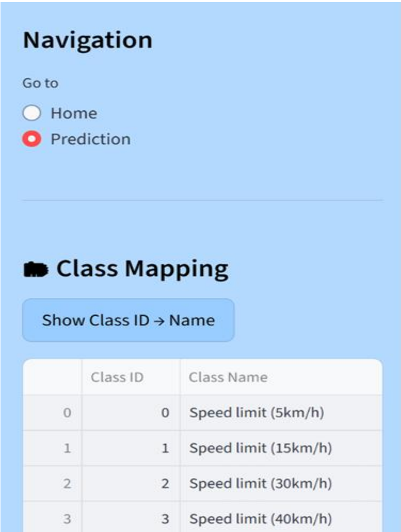
7.3 Streamlit Application

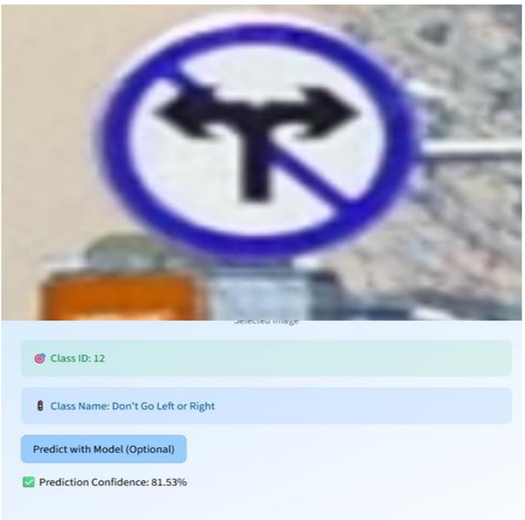
The Streamlit interface includes:

- Image upload section
- Real-time prediction output
- Probability confidence display
- Supports any traffic sign image format

Scalability options:

- Live webcam detection
- Batch processing
- Cloud hosting for global accessibility





8. Conclusion & Future Scope

8.1 Conclusion

- A CNN-based traffic sign classifier was successfully built and deployed.
- Achieved **~86% validation accuracy**, making it suitable for prototype-level real-time systems.
- The Streamlit dashboard adds strong usability and demonstrates practical deployment.

8.2 Future Enhancements

1. **Advanced Hyperparameter Tuning**
   Use Bayesian tuning, AutoML, or larger datasets.
2. **Real-Time Video Processing**
   Integrate OpenCV live stream detection.
3. **Voice-Based Feedback**
   Provide sign warnings in multiple languages.
4. **Mobile/Edge Deployment**
   Convert model to TensorFlow Lite for mobile devices.
5. **Cloud Deployment**
   Host via AWS/GCP for global accessibility.