**Road Signs Classification**

**Using Deep Learning**

**1.Context**

This project uses deep learning to classify road signs from images. With the rise of smart vehicles and traffic monitoring systems, teaching computers to recognize traffic signs has become important. The project uses a labeled dataset of traffic signs to train a model for automatic recognition.

**2.Objective**

The main aim is to build a Convolutional Neural Network (CNN) that can learn from labeled images and accurately identify the type of road sign in new images. This helps in creating systems that support safer driving.

**3.Scope**

The project covers the full cycle of a deep learning task—from loading and preprocessing the dataset, training a CNN model, evaluating its accuracy, and visualizing the results. It is limited to classifying road signs available in the given dataset.

**4.Audience**

This project is suitable for students, beginners, and developers who are learning about image classification, computer vision, or traffic-related AI systems.

**5.Techniques Used**

This project used Python and libraries like NumPy, Pandas, Matplotlib, Seaborn, and TensorFlow/Keras. Data was pre-processed through normalization and shuffling. A CNN model was trained with early stopping, and performance was measured using accuracy, confusion matrix, and classification report.

**6. Dataset Overview**

The dataset, sourced from Kaggle, is stored in three pickle files (train.p, valid.p, and test.p) containing pre-processed traffic sign images and their corresponding labels. Each image is 32×32 pixels in RGB format, and the labels are integers representing various types of road signs. A separate signnames.csv file maps each label ID to its respective traffic sign name. The dataset covers 43 unique classes of road signs. This compact and well-structured format allows for quick loading and efficient use in deep learning models.

**7. Data Cleaning**

Since the dataset is already pre-processed and stored in pickle files, minimal cleaning was required. The data was loaded and checked for missing values, duplicate entries, and label mismatches. Pixel values were normalized to improve model performance, and labels were one-hot encoded for compatibility with the deep learning model.

**8. Dataset Pre-Processing**

The dataset, sourced from Kaggle, was provided in preprocessed pickle files for training, validation, and testing. Images were normalized by dividing pixel values by 255.0 to improve model training efficiency. Labels were loaded, mapped to their respective sign names, and shuffled to avoid training bias. A quick visual check of sample images ensured correct loading and label mapping.

**9. Model Creation**

A Convolutional Neural Network (CNN) was designed with multiple convolution and pooling layers to extract spatial features from the images. The architecture included:

* Conv2D + MaxPooling2D layers for feature extraction.
* Flatten to convert feature maps into a vector.
* Dense layers for classification, ending with 43 output nodes (one for each class) with a softmax activation.

The model was compiled using the Adam optimizer, Sparse Categorical Crossentropy loss, and Accuracy as the evaluation metric.

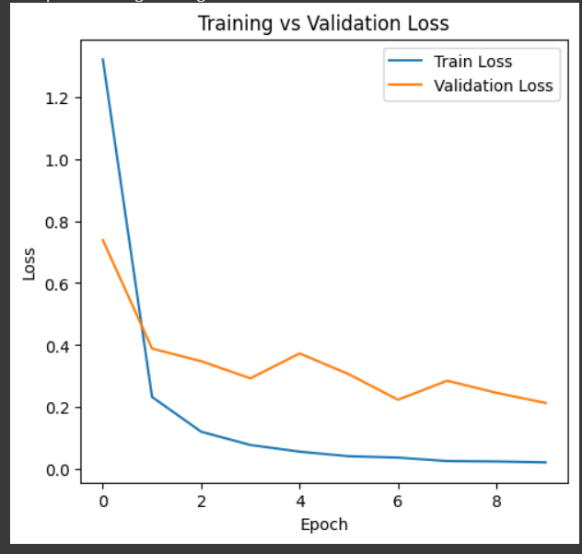
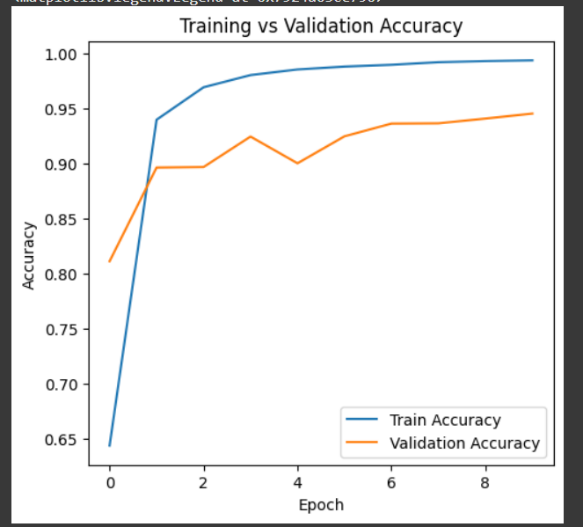
**10. Model Optimization**

To improve performance and prevent overfitting, an optimized CNN was developed with increased filter counts, Dropout layers for regularization, and EarlyStopping to halt training when validation loss stopped improving. The batch size and number of epochs were adjusted for better convergence.

**11. Model Performance**

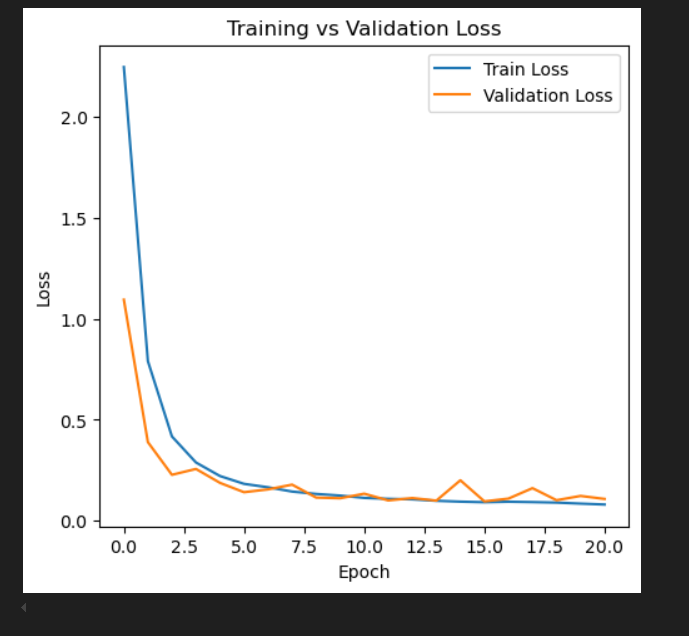
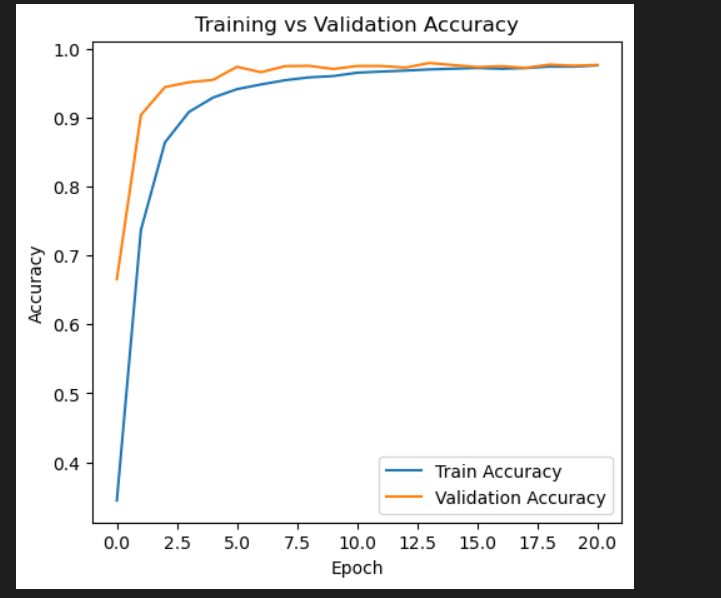
**Before Optimization**

* Training Accuracy vs Validation Accuracy  
  Initially, the model achieved high training accuracy very quickly, approaching 99% within a few epochs. However, the validation accuracy plateaued around 92–94%, showing a noticeable gap between training and validation curves. This gap suggests overfitting, where the model learned the training data patterns too well but struggled to generalize to unseen data.
* Training Loss vs Validation Loss  
  Training loss dropped rapidly to nearly zero, while validation loss decreased slower and fluctuated. These fluctuations in validation loss are another sign of overfitting, as the model’s performance on unseen data was inconsistent.



**After Optimization**

* Training Accuracy vs Validation Accuracy  
  After optimization (likely through regularization, dropout, learning rate adjustment, or data augmentation), the model achieved balanced performance. Training accuracy and validation accuracy curves were much closer, both approaching 98–99%. This indicates better generalization and less overfitting.
* Training Loss vs Validation Loss  
  Both training and validation loss decreased smoothly and converged to near-zero values without large fluctuations. This stability reflects improved model robustness and better adaptation to unseen data.

****

**12. User Interface**

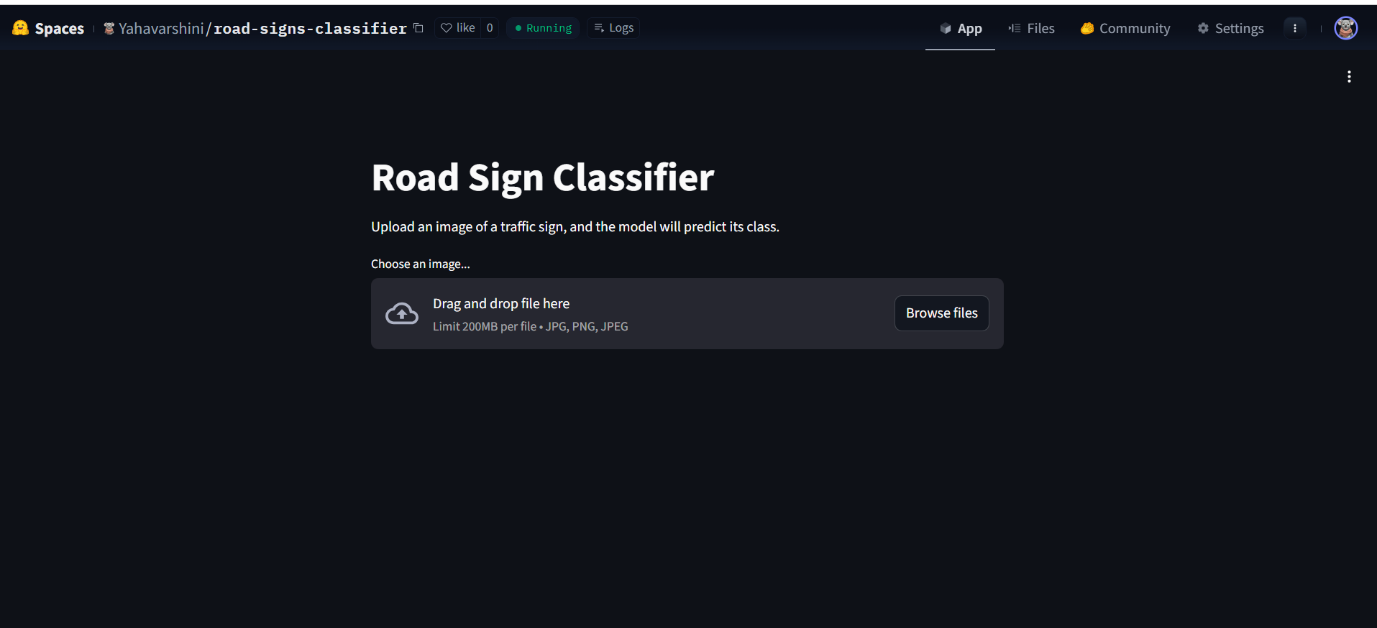
The model is deployed through a simple and interactive web application built using Streamlit. This interface allows users to upload an image of a road sign, and the model will instantly predict its class.

The trained model is stored in Hierarchical Data Format (.h5) to ensure smooth loading and integration with the web app. The UI includes:

* An image upload feature (supports .jpg, .png, and .jpeg formats).
* Real-time prediction display, showing the identified road sign label.
* A clean and user-friendly layout for easy interaction.

This interface makes the system accessible for both technical and non-technical users, enabling quick testing and demonstration of the road sign classification model.

The model is integrated into a simple and interactive Streamlit web interface, making it easy for users to upload an image of a road sign and receive instant predictions. The interface displays the predicted class along with its confidence score. This application has been deployed on Hugging Face Spaces, allowing anyone to access and test the system directly through a web browser without installing any dependencies.





**13. Conclusion**

This project successfully developed a deep learning-based Road Sign Classification system using Convolutional Neural Networks (CNN). The model was trained on a Kaggle dataset and achieved accurate predictions across multiple road sign classes. The integration with a Streamlit web application makes it easy for users to upload images and receive instant results.

The system can be extended in the future by adding more sign categories, improving accuracy with advanced architectures, and integrating it into real-time traffic systems for driver assistance and autonomous vehicles.

**14. Future Scope**

In the future, this system can be improved by training on larger and more diverse datasets to handle road signs from different countries and conditions. Advanced deep learning models like ResNet or EfficientNet can be used to improve accuracy and speed. The project can also be integrated with real-time video processing for use in autonomous vehicles and traffic monitoring systems. Additionally, support for multi-language road signs and detection in poor weather or low-light conditions can make the system more robust and practical for real-world applications.