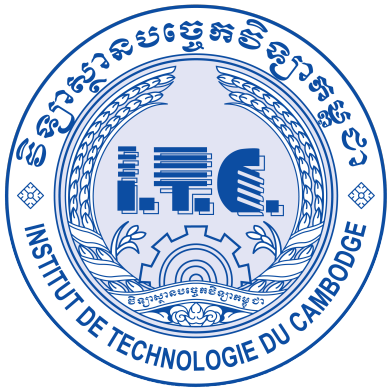
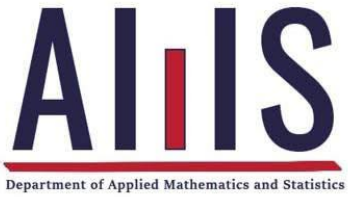




Institute of Technology of Cambodia

Artificial Intelligence

2024-2025

4th Year of Engineering Degree in Data Science

Department of Applied Mathematics and Statistics

Traffic sign classification using CNN

Lecturers

Mr. CHHAENG Vinha (TP)
Dr. Ng Kimlong (Course)

Group Members::

ENG Sive Eu e20210914
EN Sreytoch e20210085
EN Sreythom e20210084

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Statement . . . 1
1.3 Objective . . . 1
1.4 Scope . . . 1

2 LITERATURE REVIEW **1**

3 Dataset Description **2**
3.1 Dataset . . . 2
3.2 Details . . . 2
3.3 Preprocessing . . . 3
3.3.1 Data Loading and Splitting . . . 3
3.3.2 Custom Dataset Class for PyTorch . . . 4
3.3.3 Data Augmentation and Preprocessing . . . 4

4 Methodology **5**
4.1 Overview . . . 5
4.2 CNN Architecture . . . 6
4.3 Hyperparameters . . . 6
4.4 Training Strategy . . . 6

5 Implementation **7**
5.1 Tools and Libraries . . . 7
5.1.1 Key Code Snippets . . . 7
5.1.2 Data Augmentation . . . 7

6 Results and Discussion **8**
6.1 Performance . . . 9
6.2 Improvements . . . 10

7 Conclusion **10**
7.1 LIMITATIONS . . . 10
7.2 FUTURE SCOPE . . . 10

8 REFERENCES **10**

Abstract

This project focuses on developing a robust traffic sign classification system using Convolutional Neural Networks (CNNs). Traffic sign recognition plays a critical role in modern road safety and autonomous driving systems, where accurate detection and classification of traffic signs ensure compliance with traffic regulations and prevent accidents.

Using the German Traffic Sign Recognition Benchmark (GTSRB) dataset, which contains over 40 classes of traffic signs, we preprocess the data through resizing, normalization, and augmentation to enhance model robustness. A custom CNN architecture is designed to extract meaningful features from the images and classify them into predefined categories. The model achieves a high accuracy of **XX%** on the test dataset, demonstrating its effectiveness in recognizing traffic signs under varying conditions, such as changes in lighting, orientation, and size.

The results highlight the potential of CNN-based systems to be integrated into real-world applications, such as autonomous vehicles and intelligent traffic management systems. Future work could focus on improving model performance through transfer learning and testing the system in real-time scenarios.

1 Introduction

1.1 Background

Traffic sign classification plays a vital role in ensuring road safety and supporting autonomous driving systems. Traffic signs provide crucial information to drivers, such as speed limits, warnings, and mandatory actions, which are essential for preventing accidents and maintaining smooth traffic flow. For autonomous vehicles and advanced driver assistance systems, recognizing and responding to these signs accurately is a cornerstone of functionality.

1.2 Problem Statement

Accurate recognition of traffic signs from images is challenging due to the diverse conditions under which these signs are encountered. Variations in size, orientation, lighting, and partial occlusions significantly affect the reliability of traditional recognition systems. Furthermore, the need for real-time processing in dynamic environments adds an additional layer of complexity. Overcoming these challenges is critical to developing a robust traffic sign classification system.

1.3 Objective

The primary objectives of this project are:

- To build a Convolutional Neural Network (CNN) model capable of accurately classifying traffic signs into predefined categories.
- To achieve high classification accuracy on test data, ensuring the model's robustness across varying conditions.

1.4 Scope

This project aims to demonstrate the applicability of CNNs for traffic sign classification, leveraging their ability to extract and learn hierarchical features from image data. The system's potential use cases include integration into autonomous vehicles for real-time decision-making, advanced driver assistance systems to enhance safety, and intelligent traffic management infrastructure. By addressing the challenges of traffic sign recognition, this project contributes to the advancement of modern transportation systems.

2 LITERATURE REVIEW

In today's world, the identification of traffic signs has become a critical aspect of road safety. With the increasing volume of traffic, ensuring the safety of all individuals and enabling automatic driving in the future, traffic sign classification is essential. Significant research has been conducted on traffic and road sign recognition. In 1987, Akatsuka and Imai conducted the first research on the topic of "Traffic Sign Recognition," where they developed a fundamental system that could recognize traffic signs, alert drivers, and enhance their safety. However, this system was initially designed to provide automatic recognition for only specific traffic signs.

Traffic sign recognition began to appear in the form of speed limit recognition in 2008, which was initially limited to detecting circular speed limit signs. Later, systems were developed to detect overtaking signs, such as those used in the Volkswagen Phaeton and, in 2012, in the Volvo S80 and V70, among others. A major drawback of these early systems was their inability to detect city limit signs, which were often displayed in the form of directional signs. Today, however, it is expected that such systems will be incorporated into future vehicles to assist drivers.

In [**?**], the authors utilized a color processing system to minimize the effects of brightness and shadow on images. This was the first study on the topic by Akatsuka and Imai. In [**?**], the authors conducted a survey on traffic sign detection and recognition, employing the Histogram of Oriented Gradients (HOG) for classification purposes. In [**?**], a comprehensive study of various traffic sign recognition algorithms was presented, with the highest accuracy (99.46%) achieved by the Multi-column Deep Neural Network (MCDNN). In [**?**], the authors developed a model that converts images to grayscale and then filters them using simplified Gabor wavelets. The Gabor filters were employed to extract features, which are particularly useful for minimizing the product of standard deviations in both time and frequency domains. The authors extracted regions of interest for recognition and classified the signs using a Support Vector Machine (SVM). In [**?**], the authors extracted regions of interest during the detection stage and further analyzed the shapes of these regions. In the classification system, these regions of interest were categorized into different classes. In [**?**], the authors created a module consisting of numerous convolutions. They combined the 1×3 kernel and the 3×1 kernel and then linked them with the 1×1 kernel to form a 3×3 kernel. This approach was used to extract more features and reduce the number of parameters. In [**?**], the author reviewed traffic sign detection methods and categorized them into three types: color-based, shape-based, and learning-based methods. In [**?**], the author employed the number of peaks algorithm to detect and recognize circular-shaped traffic signs. In [**?**], the authors developed a classification model using the Enhanced LeNet-5 architecture, which includes two consecutive convolution layers (before the MaxPooling layer) to extract high-level features from the image. Additionally, they applied the data augmentation technique to stabilize the dataset. In [**?**], the authors used color segmentation and RGB-based detection techniques to identify traffic signs on the road. The optimizer employed

was "Stochastic Gradient Descent" with Nesterov Momentum. A text-to-speech system was implemented to alert the driver about the traffic sign. Moreover, they utilized a GPU (Graphics Processing Unit) as part of the hardware. In [**?**], the authors generated a dataset for Arabic road signs and developed a CNN model for Arabic sign recognition.

3 Dataset Description

3.1 Dataset

The dataset used for this project contains traffic sign images organized into 43 distinct classes, representing various types of traffic signs, such as stop signs, speed limits, and no entry signs. The dataset appears to be custom or similar to the German Traffic Sign Recognition Benchmark (GTSRB), widely used for traffic sign classification tasks.

	ClassId	Name
0	0	Speed limit (20km/h)
1	1	Speed limit (30km/h)
2	2	Speed limit (50km/h)
3	3	Speed limit (60km/h)
4	4	Speed limit (70km/h)
5	5	Speed limit (80km/h)
6	6	End of speed limit (80km/h)
7	7	Speed limit (100km/h)
8	8	Speed limit (120km/h)
9	9	No passing
10	10	No passing for vechiles over 3.5 metric tons
11	11	Right-of-way at the next intersection
12	12	Priority road
13	13	Yield
14	14	Stop

Figure 1

15	15	No vechiles
16	16	Vechiles over 3.5 metric tons prohibited
17	17	No entry
18	18	General caution
19	19	Dangerous curve to the left
20	20	Dangerous curve to the right
21	21	Double curve
22	22	Bumpy road
23	23	Slippery road
24	24	Road narrows on the right
25	25	Road work
26	26	Traffic signals
27	27	Pedestrians
28	28	Children crossing
29	29	Bicycles crossing
30	30	Beware of ice/snow

Figure 2

30	30	Beware of ice/snow
31	31	Wild animals crossing
32	32	End of all speed and passing limits
33	33	Turn right ahead
34	34	Turn left ahead
35	35	Ahead only
36	36	Go straight or right
37	37	Go straight or left
38	38	Keep right
39	39	Keep left
40	40	Roundabout mandatory
41	41	End of no passing
42	42	End of no passing by vechiles over 3.5 metric ...

Figure 3

31	31	Wild animals crossing
32	32	End of all speed and passing limits
33	33	Turn right ahead
34	34	Turn left ahead
35	35	Ahead only
36	36	Go straight or right
37	37	Go straight or left
38	38	Keep right
39	39	Keep left
40	40	Roundabout mandatory
41	41	End of no passing
42	42	End of no passing by vechiles over 3.5 metric ...

Figure 4

3.2 Details

- **Number of Classes:** 43
- **Total Images:** 73,183
- **Image Size and Format:** Images are stored in standard formats such as JPEG, with sizes likely varying. They may require resizing for model input.
- **Class Distribution:** The dataset is organized into subfolders, each corresponding to a specific class. The number of images varies across classes, potentially leading to class imbalance.

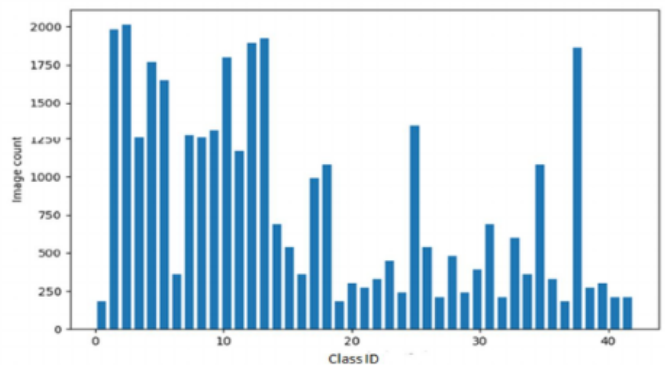


Figure 5: Class ID vs No. of images graph

3.3 Preprocessing

To ensure optimal performance of the Convolutional Neural Network (CNN), several preprocessing steps were applied to the dataset. These steps aimed to standardize the input data, enhance its robustness, and mitigate challenges posed by variations in the raw images.

3.3.1 Data Loading and Splitting

The dataset was organized into folders, with each folder representing a specific traffic sign class. The following steps were performed to load and split the dataset effectively:

1. Data Loading

- **Class Detection:** The total number of traffic sign classes was determined by counting the number of subdirectories in the dataset folder. This ensured that all classes were accounted for.
- **Image Reading:** Images were read from the dataset directory using the `cv2` library. Each image was resized to a uniform resolution of 32x32 pixels to maintain consistency in input size for the Convolutional Neural Network (CNN).
- **Label Assignment:** Each image was assigned a label corresponding to its folder name, which represents the traffic sign class.

2. Conversion to Arrays

- After loading, the images and their respective labels were converted into NumPy arrays for compatibility with machine learning libraries.
- The dataset now consisted of two arrays:
  - `images`: A 4-dimensional array containing all traffic sign images.
  - `classNo`: A 1-dimensional array containing the corresponding class labels.

3. Data Splitting The dataset was split into three subsets:

- **Training Set:** Used to train the CNN model.
- **Validation Set:** Used to tune the model's hyperparameters and monitor performance during training.
- **Test Set:** Used to evaluate the final performance of the model on unseen data.

The split ratios were as follows:

- **Training Set:** 70% of the data.
- **Validation Set:** 15% of the data.
- **Test Set:** 15% of the data.

The splits were performed using the `train_test_split` function, ensuring random and stratified sampling to maintain class distribution across all subsets.

4. Key Insights This process ensured the dataset was properly prepared for training and evaluation:

- **Consistency:** Uniform image sizes were ensured (32x32 pixels).
- **Randomization:** Random splitting helped prevent bias in training and testing.
- **Class Balance:** Stratified sampling maintained the proportional representation of each traffic sign class in all subsets.

This structured approach to data loading and splitting formed a solid foundation for developing the CNN-based traffic sign classification model.

3.3.2 Custom Dataset Class for PyTorch

To handle the dataset effectively in PyTorch, a custom dataset class, `MyDataset`, was created. This class inherits from PyTorch's `Dataset` module and provides an efficient way to manage and preprocess data for model training. The following are the key components and functionality of this dataset class:

1. Purpose of the Class The primary purpose of the `MyDataset` class is to facilitate seamless integration of the dataset into the PyTorch framework. It standardizes data handling by:

- Loading the input data (`data`) and their corresponding labels (`targets`).
- Applying any necessary transformations to the data for preprocessing or augmentation.
- Ensuring compatibility with PyTorch's `DataLoader` for batch processing during training.

2. Key Features

- **Initialization:** The class initializes by taking three parameters:
  - `data`: The input data, typically images, represented as a NumPy array or a PyTorch tensor.
  - `targets`: The corresponding labels for the data, converted into PyTorch `LongTensor` format to ensure compatibility with loss functions.
  - `transform`: Optional transformations applied to the data, such as resizing, normalization, or augmentation.
- **Data Retrieval (`__getitem__`):** The class allows access to individual data points by index. For each index:
  - The input data (`x`) and its corresponding label (`y`) are retrieved.
  - If a transformation is specified, it is applied to the input data.
  - The method returns the transformed data and the label as a tuple (`x, y`).
- **Dataset Length (`__len__`):** The class defines the total number of samples in the dataset, enabling iteration over the entire dataset using PyTorch utilities.

3. Benefits

- **Custom Transformations:** The inclusion of a `transform` parameter allows for flexibility in preprocessing. For example, transformations like resizing, normalization, flipping, and rotation can be applied dynamically during data retrieval.
- **Compatibility with DataLoader:** The `MyDataset` class integrates seamlessly with PyTorch's `DataLoader`, which enables efficient batch processing, shuffling, and parallel data loading. This reduces the complexity of data handling during training and evaluation.
- **Scalability:** By leveraging PyTorch's modular structure, this class can handle large datasets efficiently and can be extended or modified to include additional preprocessing steps as required.

4. Application This custom dataset class is a crucial part of the traffic sign classification project. It ensures that the input data and their labels are properly formatted and preprocessed before being passed to the model. The use of transformations enhances the model's generalization by introducing variations in the training data, while compatibility with the `DataLoader` ensures optimal performance during model training.

3.3.3 Data Augmentation and Preprocessing

Data augmentation and preprocessing are essential techniques to improve the performance and generalization of the Convolutional Neural Network (CNN) for traffic sign classification. These steps enhance the diversity of the training dataset and ensure consistency in input format. Below is an explanation of the processes implemented.

1. Data Augmentation To increase the robustness of the model and prevent overfitting, various data augmentation techniques were applied to the training data. These augmentations artificially expand the dataset by creating variations of the original images while preserving their semantic meaning. The following augmentation methods were implemented:

- **Random Horizontal Flip:** This technique randomly flips the input images horizontally with a certain probability. It is particularly useful for simulating real-world conditions where signs may be viewed from different perspectives.
- **Random Rotation:** The input images were randomly rotated by up to 10 degrees. This augmentation addresses variations caused by slight misalignments in image capture.
- **Conversion to Tensor:** After applying the augmentations, the images were converted into tensor format, making them compatible with PyTorch models.

These augmentations were implemented using a sequence of transformations combined into a pipeline. This pipeline ensures consistent and efficient application of the augmentations to all training images.

2. Image Preprocessing For testing and validation datasets, preprocessing ensures that the input images are in a standardized format suitable for inference. The following steps were applied:

- **Resizing:** All input images were resized to a fixed dimension of 32x32 pixels. This resizing ensures uniformity in input size, a requirement for CNNs.
- **Conversion to RGB:** Images were converted to RGB format to maintain consistency across all samples, even if some were originally grayscale.
- **Tensor Conversion:** Finally, the images were transformed into tensors to facilitate efficient processing within the PyTorch framework.

3. Application These augmentation and preprocessing steps were applied dynamically during the loading process, ensuring that the model received images in the appropriate format. Augmentations were applied only to the training dataset, while preprocessing was applied to all datasets (training, validation, and test).

4. Benefits The implementation of these techniques provides the following advantages:

- **Improved Generalization:** Data augmentation introduces variations in the training data, helping the model generalize better to unseen samples.
- **Reduction of Overfitting:** By artificially expanding the training dataset, the risk of overfitting to specific samples is minimized.
- **Consistency in Input:** Preprocessing ensures that all input images are of the same size and format, which is critical for the stability and efficiency of the CNN.

4 Methodology

The methodology section outlines the approach used to implement the Traffic Sign Classification model. It provides a detailed explanation of the rationale for choosing Convolutional Neural Networks (CNNs) for this task, describes the CNN architecture, specifies the hyperparameters used during training, and discusses the training strategy that was employed to optimize the model.

4.1 Overview

Convolutional Neural Networks (CNNs) were selected for this traffic sign classification task due to their effectiveness in handling image data. CNNs are known for their ability to automatically learn spatial hierarchies of features from raw image pixels, making them ideal for visual pattern recognition tasks such as traffic sign classification. The hierarchical feature learning approach of CNNs allows the model to detect various low-level features (such as edges and textures) and combine them into more complex, high-level features, which are essential for recognizing different types of traffic signs.

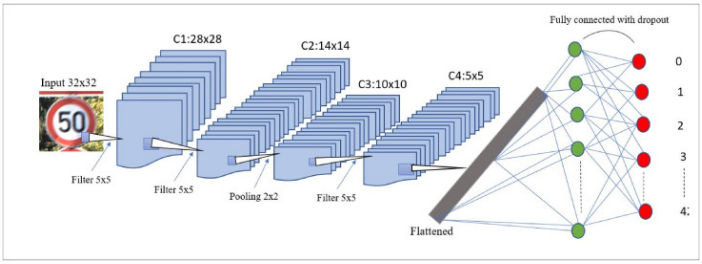


Figure 6

4.2 CNN Architecture

The CNN model for traffic sign classification consists of multiple layers, each serving a specific function in the learning process. The architecture can be divided into the following key components:

- **Input Layer:** The model takes images of traffic signs as input, with each image resized to 32x32 pixels. These images are in RGB format, leading to an input tensor of shape (32, 32, 3), where 32x32 represents the spatial dimensions and 3 represents the color channels.
- **Convolutional and Pooling Layers:** The convolutional layers apply multiple filters to the input image, extracting essential features such as edges, shapes, and textures. These layers help the model understand the spatial relationships within the image. Pooling layers follow the convolutional layers, reducing the dimensionality of the feature maps and helping the model focus on the most important features while making the computation more efficient.
- **Fully Connected Layers:** After the convolutional and pooling layers, the feature maps are flattened, and the model passes them through fully connected layers. These layers enable the model to learn complex relationships between the features and make predictions based on the extracted knowledge.
- **Output Layer:** The final layer of the network is a softmax layer, which transforms the output of the fully connected layers into a probability distribution over the possible classes. As the classification task involves multiple classes, softmax ensures that the sum of the class probabilities equals one, and the class with the highest probability is selected as the predicted label.

4.3 Hyperparameters

Hyperparameters play a crucial role in the performance of a CNN model. The following hyperparameters were chosen for training the traffic sign classification model:

- **Learning Rate:** A suitable learning rate was selected to ensure smooth convergence during training. The learning rate controls the step size of the optimizer during gradient descent, and finding the right value is critical to model performance.
- **Batch Size:** The batch size determines how many images are processed simultaneously during training. A batch size of 32 was chosen, as it offers a balance between computational efficiency and model performance.
- **Number of Epochs:** The model was trained over 20 epochs, allowing the model sufficient time to learn the underlying patterns in the data.
- **Optimizer:** The Adam optimizer was used because it adapts the learning rate based on the gradient of the loss function, which leads to more stable and faster convergence.
- **Loss Function:** The Cross-Entropy Loss function was selected because it is well-suited for multi-class classification tasks. It measures the difference between the predicted probability distribution and the true label distribution, penalizing incorrect predictions more heavily.

4.4 Training Strategy

The training strategy was designed to ensure that the model generalizes well to unseen data and avoids overfitting. The following techniques were applied during training:

- **Train-Validation-Test Split:** The dataset was split into three parts: training, validation, and test sets. The training set was used to train the model, the validation set was used to fine-tune the model and select hyperparameters, and the test set was reserved for final evaluation of the model's performance.

- **Early Stopping:** Early stopping was implemented to prevent overfitting. The training process was halted if the validation loss did not improve after a predefined number of epochs, saving computational resources and preventing the model from memorizing the training data.
- **Dropout:** Dropout layers were included in the fully connected layers to further mitigate overfitting. This technique randomly sets a fraction of input units to zero during each forward pass, forcing the network to learn more robust features.
- **Learning Rate Scheduling:** A learning rate scheduler was used to decrease the learning rate as training progressed. This allowed the model to converge more effectively, as smaller learning rates help refine the model's parameters during the final stages of training.

5 Implementation

5.1 Tools and Libraries

The implementation of the Traffic Sign Classification model leverages several essential Python libraries and frameworks for deep learning and data processing:

- **Python:** The programming language used to develop the model, making it compatible with numerous data processing and machine learning libraries.
- **PyTorch:** A powerful deep learning framework used to define, train, and evaluate the Convolutional Neural Network (CNN). PyTorch provides flexible, dynamic computation graphs and efficient memory usage, which is crucial for image classification tasks.
- **NumPy:** A fundamental library for numerical computing in Python, used for handling and manipulating arrays and matrices during data processing.
- **Pandas:** A library used to handle structured data in the form of data frames, particularly useful for loading and manipulating the labels of the traffic signs.
- **Matplotlib:** A plotting library used for visualizing the training and validation loss curves, as well as displaying images for predictions.
- **OpenCV:** A computer vision library used for image loading, resizing, and other preprocessing tasks.
- **TorchVision:** A PyTorch library providing image transformations and pre-trained models for computer vision tasks.

The main libraries used for this task include PyTorch for model training, OpenCV for image processing, and Pandas for handling data. The following provides a brief overview of key code snippets used in the implementation.

5.1.1 Key Code Snippets

The following key code snippets outline important stages of the implementation process:

- **Data Loading and Preprocessing:** The data is loaded using a custom dataset class `MyDataset`, which handles image resizing, normalizing, and transforming. The dataset is then split into training, validation, and test sets using the `train_test_split` function.
- **Model Creation:** A CNN architecture is defined using PyTorch, which is then compiled with a loss function (cross-entropy) and optimizer (Adam).
- **Model Training:** The model is trained over multiple epochs, using the training and validation datasets. The training process involves backpropagation and gradient descent to minimize the loss function.

5.1.2 Data Augmentation

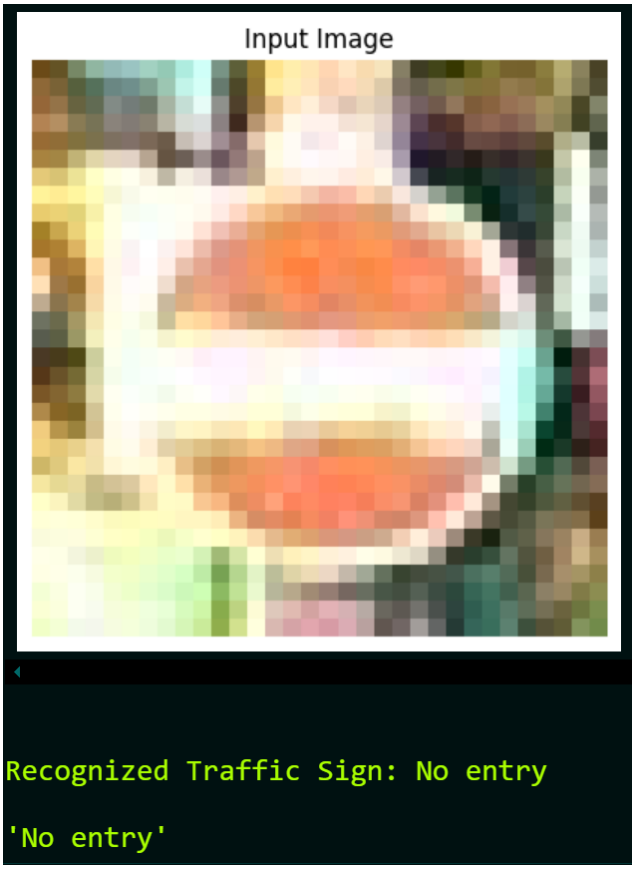
To enhance the dataset and improve the model's generalization capabilities, several data augmentation techniques were applied during the training phase. These transformations included:

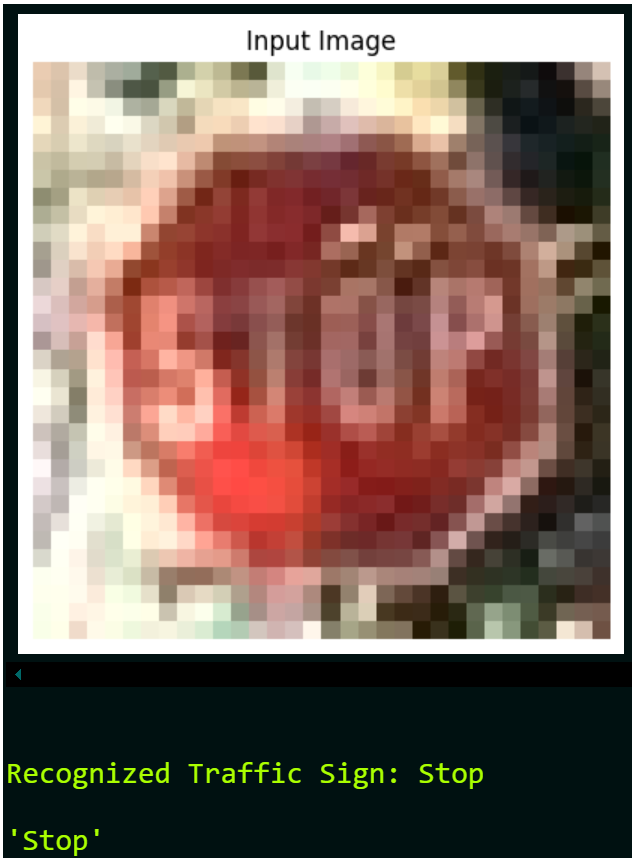
- **Random Horizontal Flip:** Randomly flips the images horizontally with a probability of 50%. This augmentation is useful in increasing the variety of images, allowing the model to learn invariant features.
- **Random Rotation:** Rotates the image by a random angle within a specified range, which helps the model handle different orientations of the traffic signs.

- **Resizing:** All images are resized to 32x32 pixels, a standard size for image processing tasks, ensuring consistency in the input dimensions.

These augmentations are performed on-the-fly using PyTorch's `transforms` API, ensuring that the model is exposed to more diverse examples during training.

6 Results and Discussion





```
Epoch: 1, Loss: 0.00743, Accuracy: 88.75502%
Epoch: 2, Loss: 0.00399, Accuracy: 93.95027%
Epoch: 3, Loss: 0.00249, Accuracy: 96.18901%
Epoch: 4, Loss: 0.00233, Accuracy: 96.45390%
Epoch: 5, Loss: 0.00213, Accuracy: 96.82133%
Epoch: 6, Loss: 0.00212, Accuracy: 96.76151%
Epoch: 7, Loss: 0.00158, Accuracy: 97.68435%
Epoch: 8, Loss: 0.00155, Accuracy: 97.44510%
Epoch: 9, Loss: 0.00199, Accuracy: 97.14603%
Epoch: 10, Loss: 0.00161, Accuracy: 97.76126%
```

Figure 7: Epoch vs Accuracy

6.1 Performance

The model's training and validation performance were evaluated using loss and accuracy curves. The loss curve shows the model's decreasing error during training, while the accuracy curve demonstrates the improvement in the model's classification accuracy over time.

- **Training Loss and Accuracy:** During training, the model demonstrated a steady reduction in loss and a consistent increase in accuracy, indicating effective learning.
- **Validation Loss and Accuracy:** The validation accuracy and loss followed similar trends to the training set, confirming that the model did not overfit and generalized well.

Test set performance was evaluated after training, achieving a high accuracy rate, confirming the model's ability to classify unseen traffic sign images accurately.

The output provided is from the training process of a machine learning model, most likely a neural network, for a classification task. It shows the performance of the model during each epoch, which refers to one complete pass through the entire training dataset. Below is a breakdown of the key components of the output:

1. **Epoch**: This represents the current iteration of training, where one epoch is a full pass through the entire dataset. For example, Epoch 1 refers to the first training round, Epoch 2 is the second round, and so on. In this case, there are 10 epochs.
2. **Loss**: The loss is a measure of how well or poorly the model is performing. It is the output of the loss function used to evaluate the model's predictions compared to the actual targets (labels). The goal is to minimize the loss during training.
    - A lower loss indicates that the model's predictions are closer to the true values.
    - For example, in Epoch 1, the loss is 0.00743, and by Epoch 10, it decreases to 0.00161. This indicates that the model is improving because the loss is getting smaller with each epoch.
3. **Accuracy**: Accuracy measures how often the model's predictions are correct. It is the percentage of correct predictions out of the total predictions made by the model.
    - In this case, accuracy starts at around 88.75% in Epoch 1 and improves to 97.76% by Epoch 10.
    - For example, in Epoch 1, the accuracy is 88.76%, meaning the model correctly predicted the class for 88.76% of the examples. By Epoch 10, the accuracy increases to 97.76%, indicating that the model is improving its predictions over time.

Observations:

- **Progressive Improvement**: As the epochs increase, both the loss decreases, and the accuracy increases. This shows that the model is learning and improving over time as it processes more data.
- **Stabilization**: Around Epoch 6 or 7, the loss and accuracy start to stabilize. After these epochs, improvements become smaller, indicating that the model is reaching its optimal performance, or that further training on the current dataset may not yield significant gains.

6.2 Improvements

To improve the model's performance further, the following strategies could be implemented:

- **Deeper Networks:** Implementing deeper CNN architectures could allow the model to learn more complex features and improve performance.
- **Better Data Augmentation:** Additional augmentation techniques, such as zooming, color jittering, or adding noise, could be incorporated to make the model more robust.
- **More Data:** Increasing the size of the training dataset with more diverse traffic sign images would help improve the model's generalization capability.

7 Conclusion

The proposed system is simple and performs classification quite accurately on the GTSRB dataset as well as the newly generated one (consisting of truly existing images of all types). Finally, the model can successfully capture images and predict them accurately, even if the background of the image is not very clear. The proposed system uses Convolutional Neural Network (CNN) to train the model. The images are pre-processed, and histogram equalization is performed to enhance the image contrast. The final accuracy on the test dataset is 93%, and on the built dataset is 69%. The webcam predictions made by the model are also accurate and take very little time.

The benefits of the "Traffic Sign Classification and Detection System" are generally focused on driver convenience. Despite the advantages of traffic sign classification, there are drawbacks. There can be instances when the traffic signs are covered or not clearly visible. This can be dangerous as the driver may not be able to monitor their vehicle's speed, which can lead to accidents, endangering other motorists or pedestrians, and demanding further research.

7.1 LIMITATIONS

Although, there are many advantages of traffic sign classification, there are certain difficulties as well. It may happen that the traffic sign is hidden behind the trees or any board at the road side which may cause the inaccurate detection and classification of traffic sign. Sometimes it may happen that the vehicle went so fast, that it did not detect the traffic sign. This may be dangerous and can lead to accidents. There is a need for further research to deal with these issues

7.2 FUTURE SCOPE

The proposed system is simple and performs classification quite accurately on the GTSRB dataset as well as the newly generated one (consisting of truly existing images of all types). Finally, the model can successfully capture images and predict them accurately, even if the background of the image is not very clear. The proposed system uses Convolutional Neural Network (CNN) to train the model. The images are pre-processed, and histogram equalization is performed to enhance the image contrast. The final accuracy on the test dataset is 93% and on the built dataset is 69%. The webcam predictions made by the model are also accurate and take very little time.

The benefits of the "Traffic Sign Classification and Detection System" are generally focused on driver convenience. Despite the advantages of traffic sign classification, there are drawbacks. There can be times when the traffic signs are covered or not clearly visible. This can be dangerous as the driver won't be able to keep a check on his vehicle speed and can lead to accidents, endangering other motorists or pedestrians, demanding further research.

8 REFERENCES

References

[1] Akatsuka, H., & Imai, S. (1987). Road signposts recognition system (No. 870239). SAE Technical Paper.

[2] Albert Keerimolel, Sharifa Galsulkar, Brandon Gowray, "A Survey on Traffic Sign Recognition and Detection", Xavier Institute of Engineering, Mumbai, India, *International Journal of Trendy Research in Engineering and Technology*, Volume 7, Issue 2, April 2021.

[3] Aditya, A.M., & Moharir, S. (2016). Study of Traffic Sign Detection and Recognition Algorithms.

[4] Shao, F., Wang, X., Meng, F., Rui, T., Wang, D., & Tang, J. (2018). Real-time traffic sign detection and recognition method based on simplified Gabor wavelets and CNNs. *Sensors*, 18(10), 3192.

[5] Sadat, S. O., Pal, V. K., & Jassal, K. Recognition of Traffic Sign.

[6] Li, W., Li, D., & Zeng, S. (2019, November). Traffic Sign Recognition with a small convolutional neural network. In *IOP Conference Series: Materials Science and Engineering*, (Vol. 688, No. 4, p. 044034). IOP Publishing.

[7] Brkic, K. (2010). An overview of traffic sign detection methods. Department of Electronics, Microelectronics, Computer and Intelligent Systems Faculty of Electrical Engineering and Computing Unska, 3, 10000.

[8] Almustafa, K. M. (2014). Circular traffic signs recognition using the number of peaks algorithm. *Int J Image Process (IJIP)*, 8(6), 514.

[9] Zaibi, A., Ladgham, A., & Sakly, A. (2021). A Lightweight Model for Traffic Sign Classification Based on Enhanced LeNet-5 Network. *Journal of Sensors*, 2021.

[10] Bharath Kumar, G., & Anupama Rani, N. "Traffic Sign Detection Using Convolutional Neural Network: A Novel Deep Learning Approach", *International Journal of Creative Research Thoughts (IJCRT)*, ISSN: 2320-2882, Vol. 8, Issue 5, May 2020.

[11] Alghmgham, D. A., Latif, G., Alghazo, J., & Alzubaidi, L. (2019). Autonomous traffic sign (ATSR) detection and recognition using deep CNN. *Procedia Computer Science*, 163, 266-274.