Classifying Traffic Signs with Convolutional Neural Networks

Sammantha Firestone

COMP4449: Capstone | Midterm Project

Dr. Claudio Delrieux

Summer Quarter 2023

**Contents**

[Classifying Traffic Signs with Convolutional Neural Networks 2](#_Toc139735901)

[Significance 2](#_Toc139735902)

[Overall Goal 3](#_Toc139735903)

[Data Source 3](#_Toc139735904)

[Data Exploration and Preparation 3](#_Toc139735905)

[Data Description 3](#_Toc139735906)

[Exploratory Data Analysis and Visualization 5](#_Toc139735907)

[Data Preprocessing 8](#_Toc139735908)

[Model Building and Tuning 10](#_Toc139735909)

[Model Building 10](#_Toc139735910)

[Model Evaluation 14](#_Toc139735911)

[Conclusion 16](#_Toc139735912)

[Appendix 17](#_Toc139735913)

[References 20](#_Toc139735914)

# 

# Classifying Traffic Signs with Convolutional Neural Networks

## Significance

Autonomous vehicles are revolutionizing the transportation industry, by leveraging advanced technology to navigate roads while making independent decisions. As essential aspect of autonomous driving is accurately detecting and classifying the world around them. This is true of traffic signs as they provide critical information for safe and efficient driving. By correctly identifying traffic signs in real-life conditions, autonomous vehicles can comply with road regulations and ensure the safety of passengers, other drivers, and pedestrians.

## Overall Goal

The overall goal is to develop a Convolutional Neural Network (CNN) model that is accurately able to classify images of traffic signs in real-life conditions.

## Data Source

The data files used are from the [Data Science Capstone repo](https://github.com/emmanueliarussi/DataScienceCapstone/tree/master/3_MidtermProjects/ProjectRTS/data) [1]. The data originally came from INI Benchmark website, labeled as [The German Traffic Sign Detection Benchmark](https://benchmark.ini.rub.de/gtsdb_dataset.html) [2]. The images in this dataset were captured driving on public roads using a vehicle equipped with a camera system. The images were later annotated by the researchers.

# Data Exploration and Preparation

## Data Description

There are a total of 51,839 traffic sign images collected for classification and these images were already split into training (12,630 images), validation (34,799 images), and testing (4,410) datasets. Each dataset was imported in as a dictionary, with the key values *coords*, *labels*, *features*, and *sizes*. *Coords* refer to the coordinates of the bounding box around the sign in the image. *Labels* refer to the label ID associated with each image that can be mapped to the sign name. *Features* refer to a 4D array containing the image data. *Sizes* refer to the width and height of the original image. For the classification, only *features* and *labels* are used as the respective input variables and target variables.

The input variable *features* contains the image index, width and height of the image, and the respective red-green-blue (RGB) pixel values. Each image within the Traffic Sign dataset is 32 by 32 pixels.

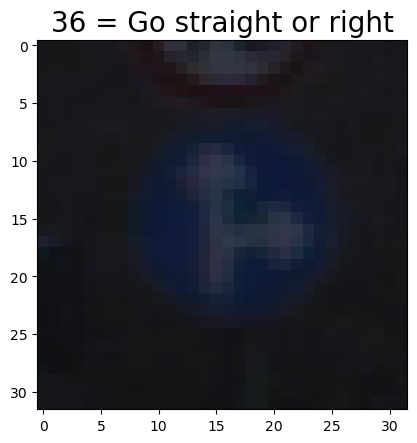
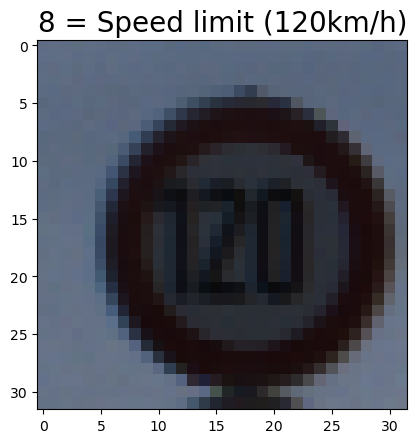


Figure 1: Sample Images from Traffic Sign Dataset

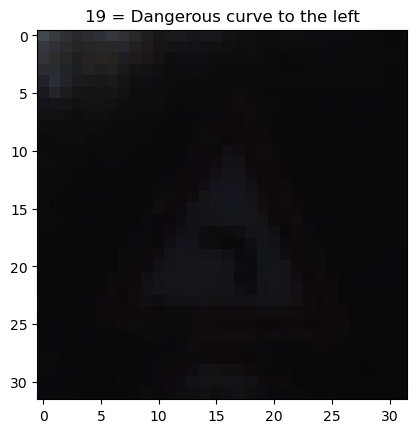
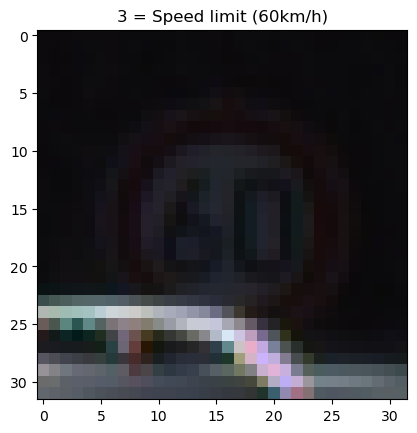
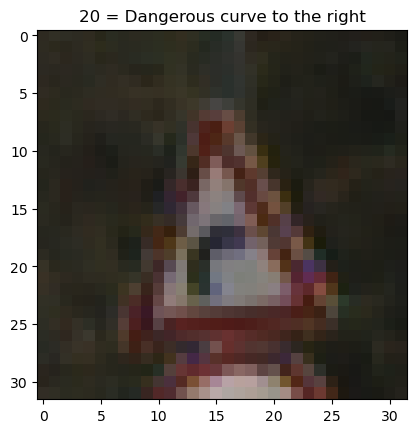
 Images are collected in real-life conditions which means that traffic signs may be obstructed by other objects, dark and noise, and / or different sizes and orientations.

Figure 2: Messy Traffic Sign Images

The target variable *labels* contain 43 unique traffic signs to be classified.



Figure 3: All Types of Traffic Signs to be Classified

## Exploratory Data Analysis and Visualization

To understand how clean or messy images were for each traffic sign label, the pixel intensity distribution was plotting for each class. Traffic sign labels that are “messy” i.e., have low visibility and noise within the majority of images, had skewed pixel intensity distribution plot. While traffic sign labels that are “clean” i.e., have clear visibility and low noise within a majority of images, had an even distribution pixel intensity distribution plot. This is because the more contrast an image has (the more even distribution of pixel intensity values), the more features there are to distinguish within the image [3]. It is important to note that these distributions are not representative of all images within that traffic class label, but an overview to understand what classes the CNN model might struggle with. Please refer to the Jupyter Notebook for further images and details.

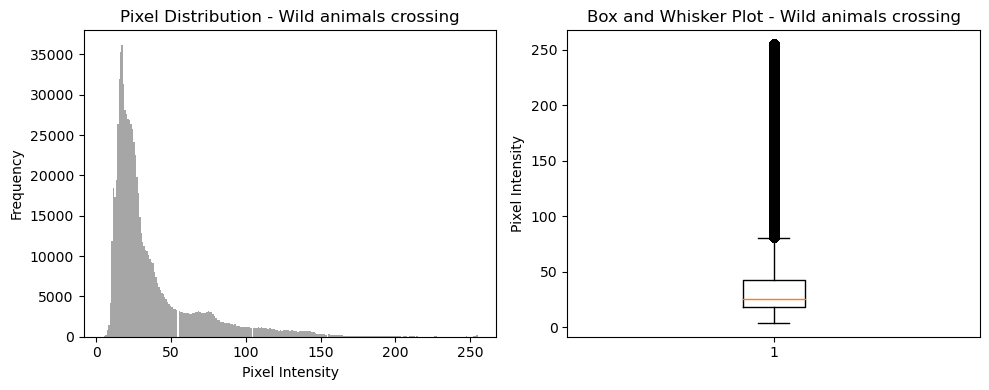


Figure 4: "Messy" Traffic Sign Class

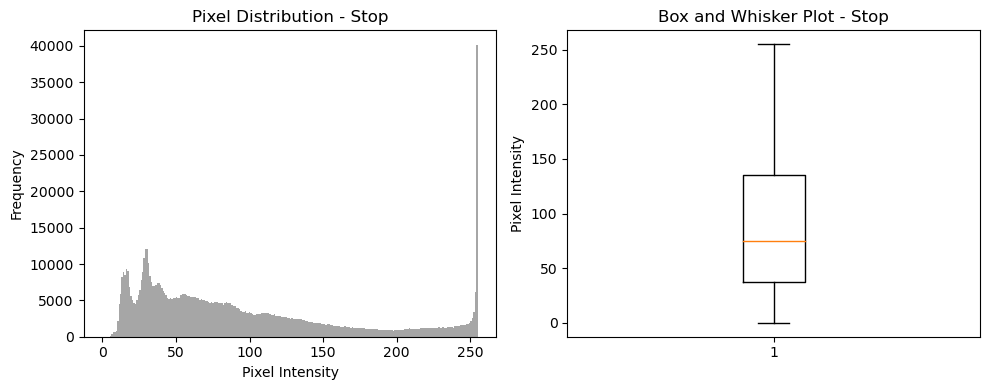


Figure 5: "Clean" Traffic Sign Class

To understand the distribution of the target variable labels, or traffic sign names, the number of images per class is plotted for the training, validation, and testing datasets.

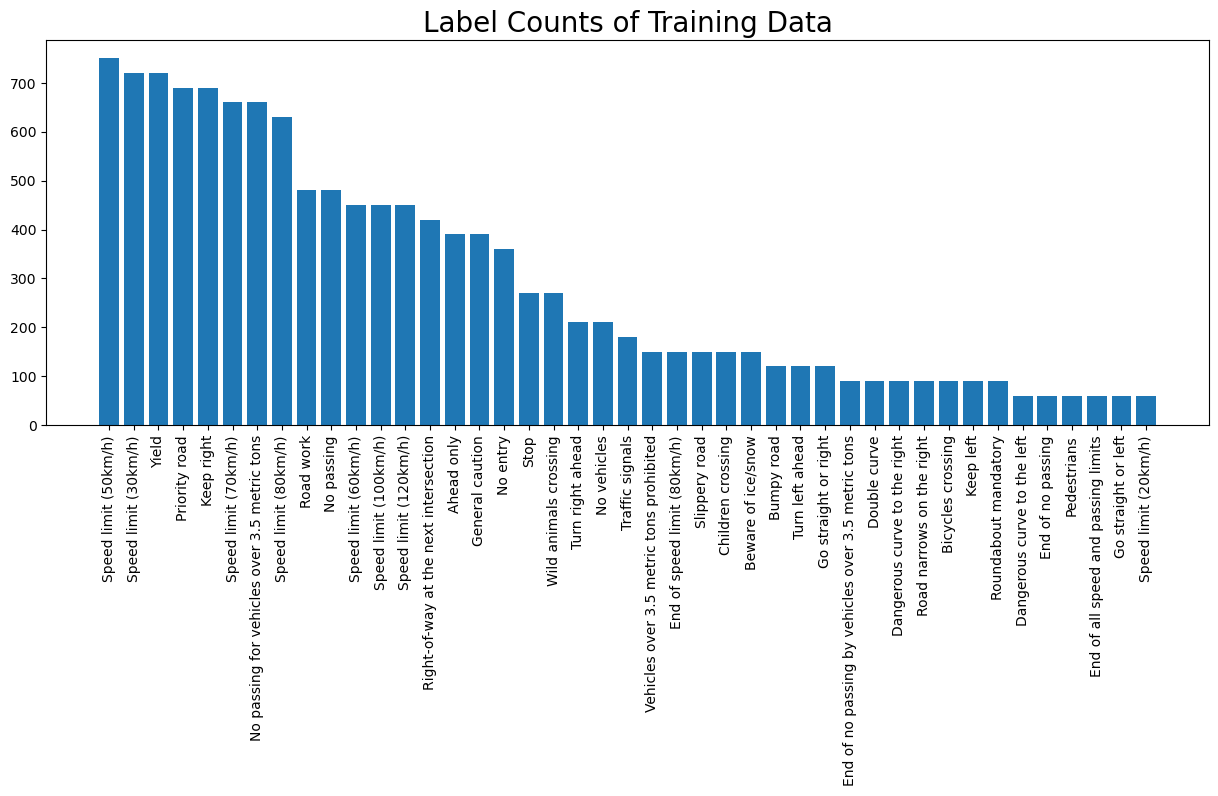


Figure 6: Image Counts per Label on Training

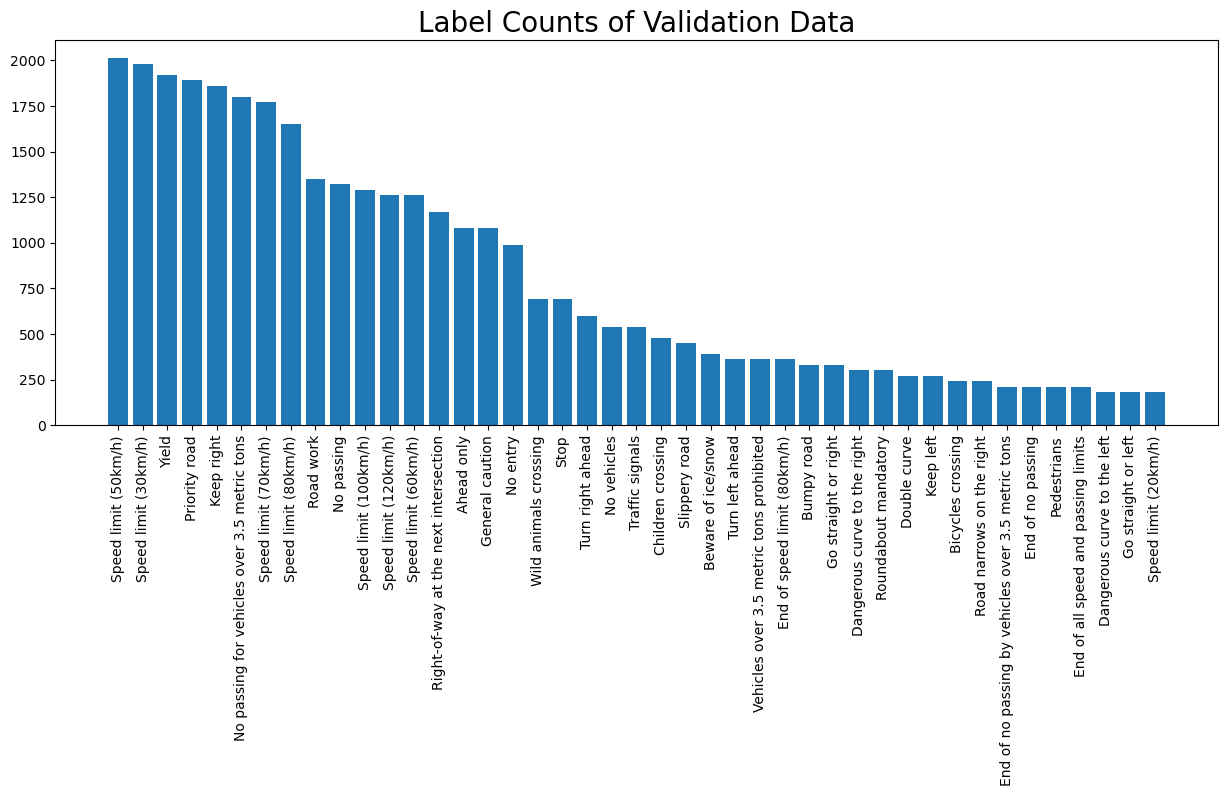


Figure 7: Image Counts per Label on Validation

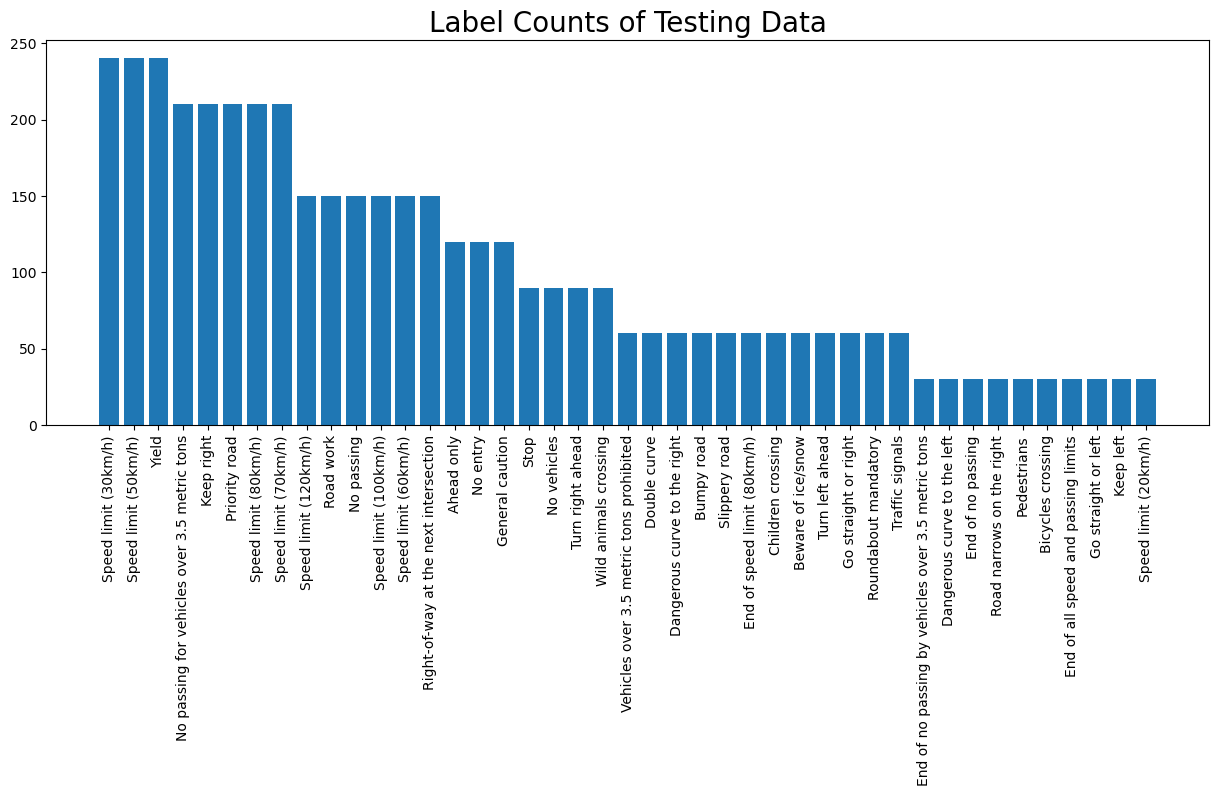


Figure 8: Image Counts per Label on Testing

All three datasets are highly skewed, with a majority of the image count in the *Speed Limit (30km/h)*, *Speed Limit (50km/h)*, and *Yield* traffic sign labels, and minority in the *Go Straight or left*, *Keep Left*, and *Speed Limit (20km/h)* traffic sign labels. This skew will create biases in the classification model towards the majority classes and will be addressed in the data preprocessing section.

## Data Preprocessing

To prepare the data for the CNN model, all images in the training, validation, and testing datasets were normalized to [0,1] from [0,255]. Normalizing the data helps increase numeric stability as well as an increase learning speed [4]. The training data is also shuffled to ensure no artificial ordering is imposed, which helps prevent bias and overfitting [5].

To address the highly skewed traffic labels on the training dataset, random over sampling was performed on any of the traffic sign labels that had less than 300 images associated with it. Random oversampling selects random examples from the minority class and with replacement, adds them to the training dataset [6]. The minority threshold value of 300 was found through trial and error.

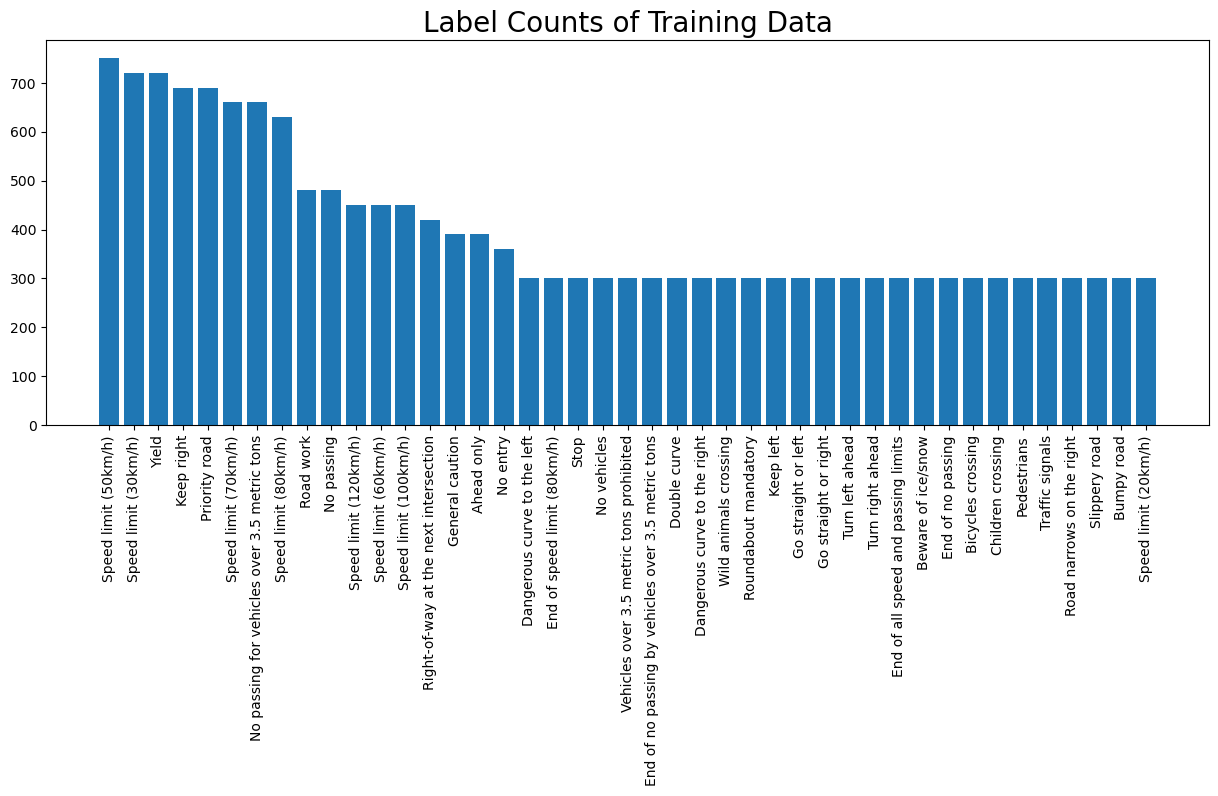


Figure 9: Image Counts per Label on Training after Random Oversampling

The traffic sign labels were one-hot encoded, converted from a single numeric column to multiple binary columns, and converted to EagerTensor data type. Using one-hot encoded target values, improves model performance and help avoids the problem of ordinality [7]. All traffic sign images in all three datasets were also converted to EagerTensor data type, as the TensorFlow library is used to build and train the CNN.

# Model Building and Tuning

## Model Building

Two Convolutional Neural Networks (CNNs) were trained to classify traffic signs to understand the impact of hyperparameters on CNN performance. The first model implement is a basic CNN [8] and the second model implement is a LeNet5 based CNN [9].

|  |  |  |
| --- | --- | --- |
|  | **Basic CNN: `cnn\_baseline`** | **LeNet-5 Based CNN: `lenet\_baseline`** |
| ***Layer 1*** | **Conv2D (32 filters, kernel size 3x3, stride 1x1,**  **'valid' padding, ReLU activation)** | **Conv2D (6 filters, kernel size 5x5, stride 1x1,**  **'same' padding, tanh activation)** |
| ***Layer 2*** | **MaxPooling2D (pool size 2x2)** | **AveragePooling2D (pool size 2x2)** |
| ***Layer 3*** | **Conv2D (32 filters, kernel size 3x3,**  **'valid' padding, ReLU activation)** | **Conv2D (16 filters, kernel size 5x5,**  **'valid' padding, tanh activation)** |
| ***Layer 4*** | **MaxPooling2D (pool size 2x2)** | **AveragePooling2D (pool size 2x2)** |
| ***Layer 5*** | **Flatten** | **Conv2D (120 filters, kernel size 5x5,**  **'valid' padding, tanh activation)** |
| ***Layer 6*** | **Dense (64 units, ReLU activation)** | **Flatten** |
| ***Layer 7*** | **Dense (43 units, softmax activation)** | **Dense (84 units, tanh activation)** |
| ***Layer 8*** |  | **Dense (43 units, softmax activation)** |

Figure 10: CNN Model Architecture

Both, the basic CNN and LetNet5 based CNN, models are trained using the training and validation datasets. To evaluate model performance, categorical accuracy, and the categorical cross-entropy (loss) is plotted and printed for both the training and validation datasets. Each model, basic CNN and LeNet5 CNN, were trained over 20 epochs with a batch size of 64. These values were chosen through trial and error, to ensure convergence and speed of the model. For further information on CNNs, reasons behind hyperparameters, and general further information, please refer to the Jupyter Notebook.

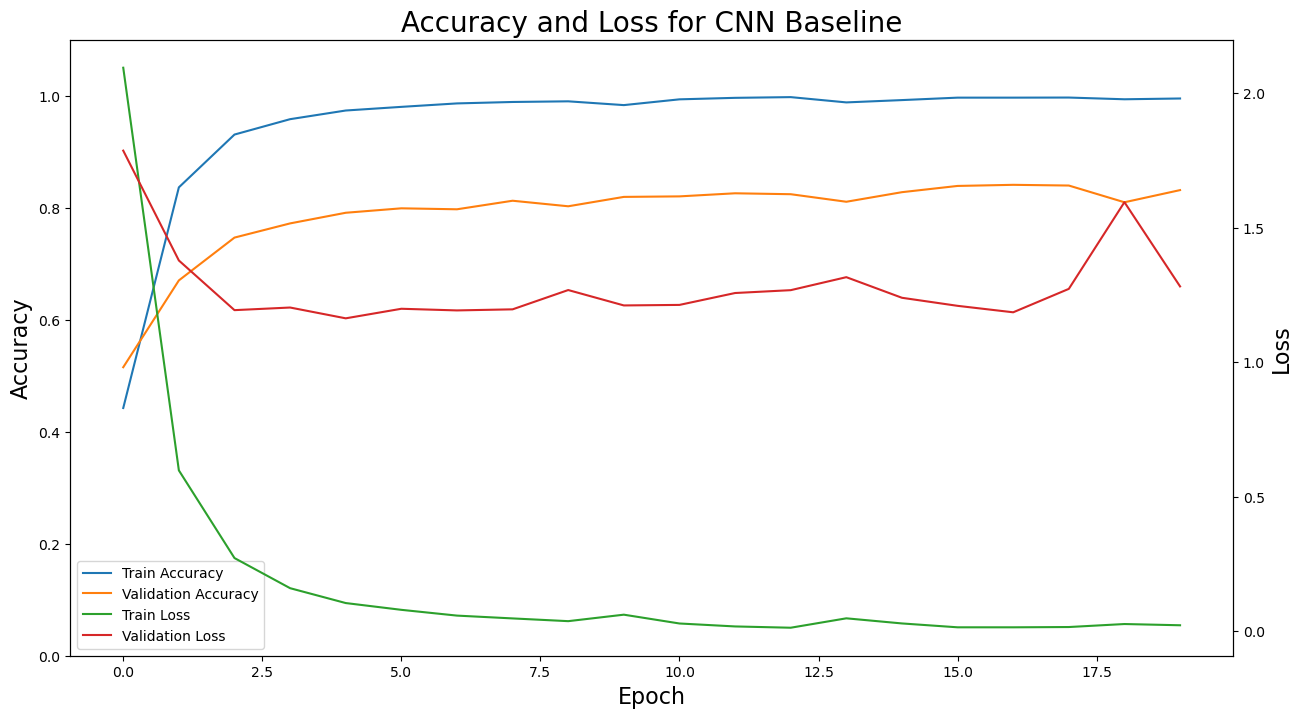


Figure 11: Basic CNN Training and Validation Accuracy & Loss Curves

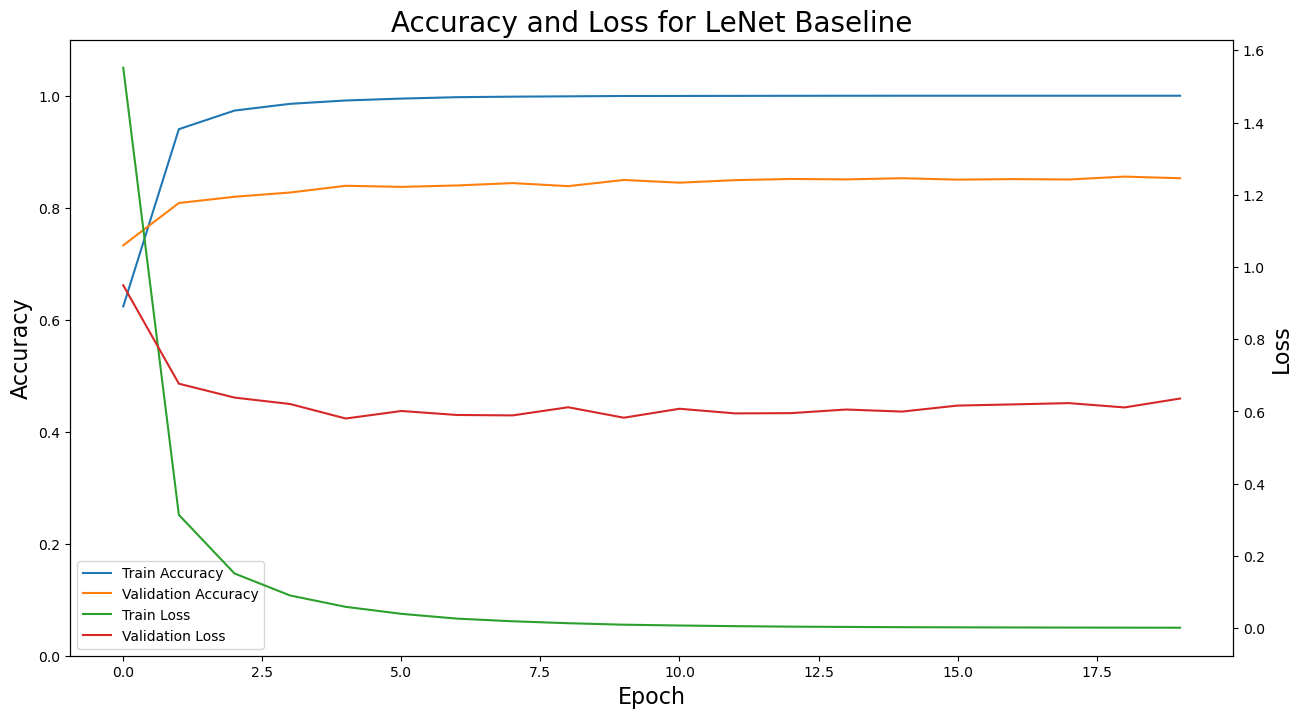


Figure 12: LeNet5 Based CNN Training and Validation Accuracy & Loss Curves

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Training Categorical Accuracy | Validation Categorical Accuracy | Training Loss | Validation Loss | Execution Time (minutes) |
| Basic CNN | 99.01% | 84.41% | 0.0364 | 1.0879 | 2.19 |
| LeNet5 CNN | 100.00% | 84.72% | 0.0011 | 0.6386 | 2.59 |

Figure 13: Training results for both CNN Models

As can be seen in *Figure 11* and *Figure 12*, both models are overfit to the training data. This can be determined as the categorical accuracy for the training data is ~100%. Also, the loss curve for the validation data is significantly higher than that of the training data.

To combat overfitting and improve overall model performance of the validation data, ridge regularization is implemented to the loss functions in the neural networks, and dropout layers are added in between each dense layer of the models. Ridge regularization is a technique to reduce overfitting in machine learning models. It does this by adding a penalty to the loss function [10]. Dropout is a technique that is also used to reduce overfitting, and it does this by randomly dropping neurons during training. The purpose of this is to force the neural network to learn with different neurons during each epoch. This prevents the neural network from relying on one neuron to lean the features of the data [11]. Both of these techniques are only implemented during training and removed during testing.

The L2 (ridge) regularization penalty used in both models is 0.001, and is applied at each dense layer of the model. Dropout layers are added between each of these dense layers as well, with 40% neurons dropped at each layer.

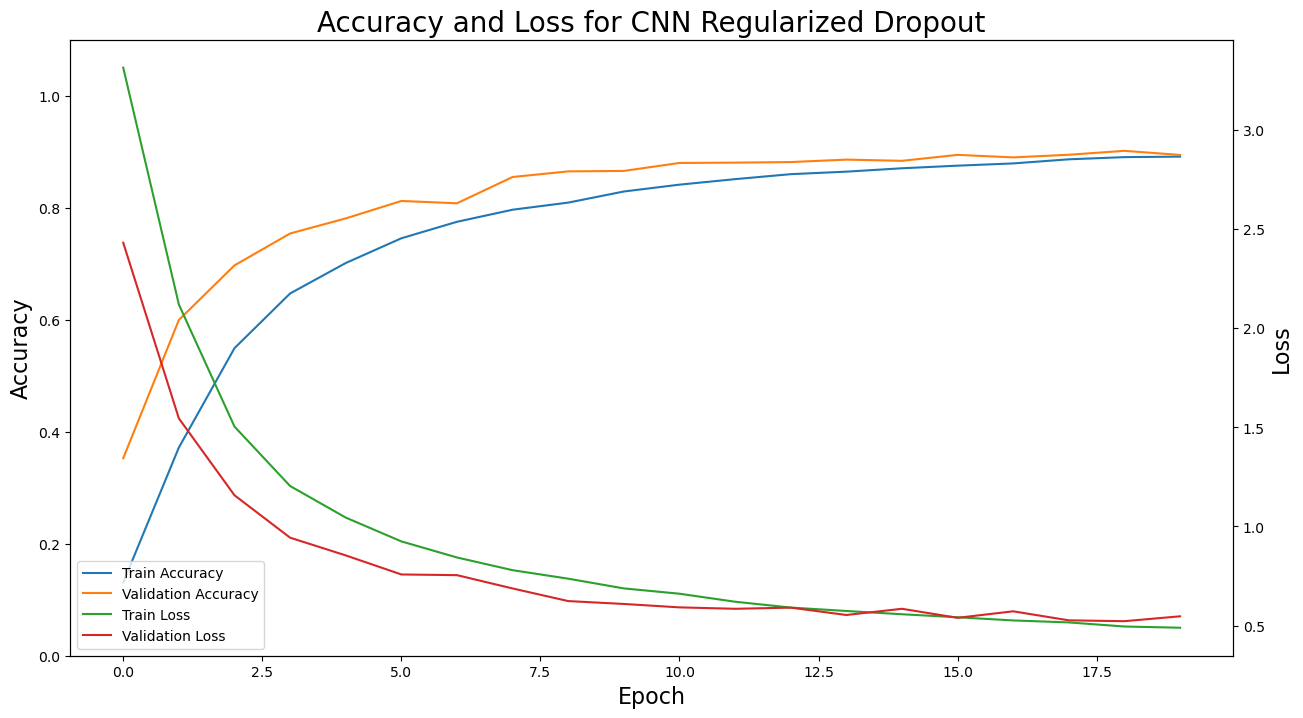


Figure 14: Basic CNN with Ridge Regularization and Dropout Layers Training and Validation Accuracy & Loss Curves

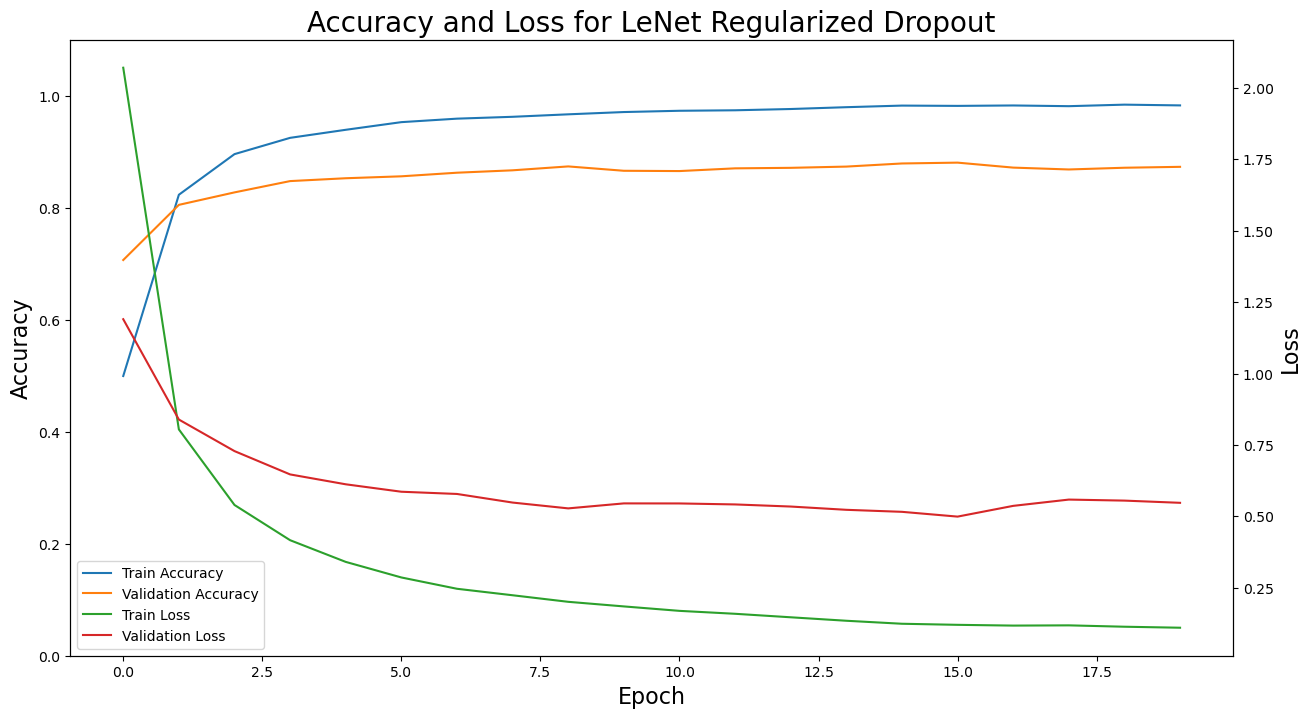


Figure 15: LeNet5 CNN with Ridge Regularization and Dropout Layers Training and Validation Accuracy & Loss Curves

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Training Categorical Accuracy | Validation Categorical Accuracy | Training Loss | Validation Loss | Execution Time (minutes) |
| Basic CNN | 88.55% | 88.12% | 0.5017 | 0.6198 | 2.31 |
| LeNet5 CNN | 98.59% | 87.70% | 0.1037 | 0.5243 | 2.28 |

Figure 16: Training results for both CNN Models after Ridge Regularization and Dropout Layers

With the addition of ridge regularization and dropout layers, both models significantly improve. There is an 4.71% categorical accuracy increase for the Basic CNN model, and an 2.98% increase for the LeNet5 CNN model.

*Figure 14* shows the performance of the Basic CNN, where the categorical accuracy curve for the validation data is slightly higher than the training data during early epochs. This is expected based on the additions made to the model. After ~12 epochs, both the training and validation accuracy and loss curves are within alignment with each other which is ideal.

*Figure 15* shows the performance of the LeNet5 CNN, where the categorical accuracy curve for the validation data is only slightly higher than the training data during first epoch. After the first epoch, the categorical accuracy becomes higher for the training data than the validation data. The loss curve shows a similar trend, where the validation loss is only lower than the training loss for the first epoch. These are all indications that the LeNet5 CNN model is still slightly overfit to the training data.

## Model Evaluation

|  |  |  |
| --- | --- | --- |
| Model | Test Categorical Accuracy | Testing Loss |
| Basic CNN with Regularization and Dropout | 90.07% | 0.4851 |
| LeNet5 CNN with Regularization and Dropout | 88.78% | 0.5044 |

Figure 17: Results for both models on testing dataset

Both models are evaluated on the testing dataset with good categorical accuracy and loss. The Basic CNN model with ridge regularization and dropout layers slightly outperform the LeNet5 based CNN model with ridge regularization and dropout layers. Because the LeNet5 model was deemed still slight overfit in *Figure 15*, it is unsurprising that it did not perform better than the good fit Basic CNN model.

Confusion matrices are plotted for both models to gain an understanding of the types of traffic signs each model struggles to correctly classify (please refer to the Appendix section for confusion matrix plots). The Basic CNN model misclassified *Speed Limit (20km/h)*, *End of no passing*, *Roundabout Mandatory*, and *Go straight or right* traffic signs the most. The LeNet5 model misclassified *Speed Limit (120 km/h)*, *Pedestrians*, *Road work*, and *Double curve* traffic signs the most. Each model had trouble with different types of traffic signs which reinforces the fact that these two models utilize different hyperparameters in their respective models.

Activation heatmaps for the first convolutional layer are plotted for both models on two different images—one image that the Basic CNN misclassified and one image that the LeNet5 Based CNN misclassified. The activation heatmap is implemented using Grad-CAM (Gradient-weighted Class Activation Mapping). This is a visualization technique that highlights the important regions in an input image that contribute the most to the predictions made by a layer within a CNN [12]. Image 142, *Go straight or right*, was misclassified by the Basic CNN model, but correctly classified by the LeNet5 Based Model.

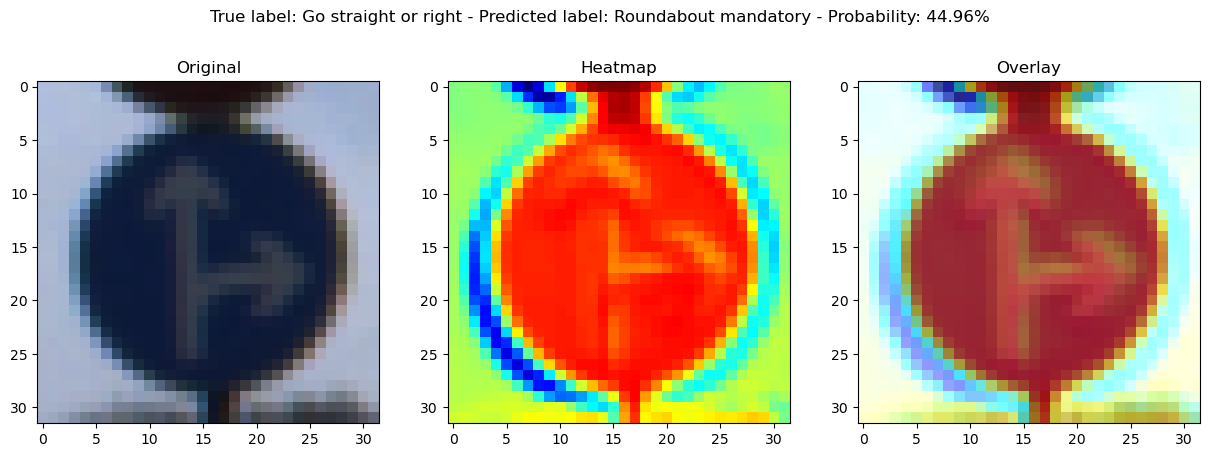


Figure 18: Basic CNN: Activation heatmap for image 142

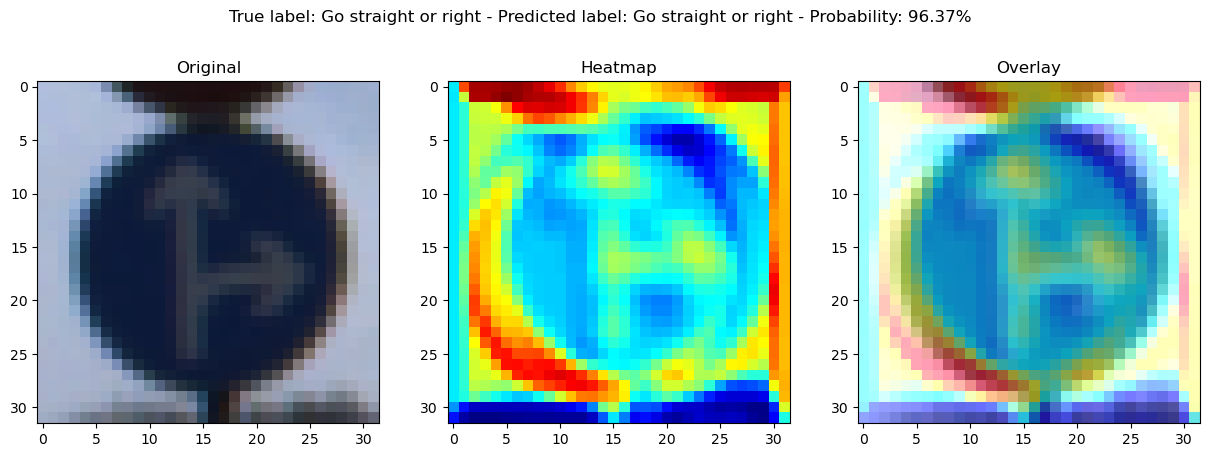


Figure 19: LeNet5 Based CNN: Activation heatmap for image 142

The heatmap in *Figure 18*, shows at the first convolutional layer of the Basic CNN model, edges of the traffic sign are found, however, it lacks definition around the arrows within the sign. The heatmap in *Figure 19*, shows at the first convolutional layer of the LeNet5 Based Model, both the edges of the traffic sign and the arrows within the sign itself are found. This visualization makes it interpretable why one model could predict the sign but not the other. Activation heatmaps for image 1342, *Speed limit (20km/h)*, can be found in the Appendix section. This image was correctly classified by the LeNet5 Based model, but misclassified for the Basic CNN model.

# Conclusion

In summary, a Convolutional Neural Network (CNN) was built to classify traffic sign images from real-life with ~90% accuracy. This required the normalization of all three datasets, and the random oversampling of minority class images in the training dataset. Then two different model architectures were used to evaluate the performance of different CNN hyperparameters given the input data. Further tuning of models were performed by adding ridge regularization to the loss functions of the neural networks, and dropout layers between each dense layer. Further investigation was done after model evaluation on the test data, to understand the key features and patterns each model determined to be most important in classification. This was done through identifying images that were misclassified using confusion matrices, and pattern detection in different convolutional layers using activation heatmaps.

It is important to note, that 90% accuracy is not high enough to be used in real world applications. To improve model performance, further steps must be done at all stages of model building. Further data preprocessing steps can be performed, such as removing a percentage of distorted or “messy” images from the training dataset and adding additional random noise to the random oversampled images. Further model tuning must be performed, ideally utilizing a grid search technique. It would be also beneficial to research and utilize alternative CNN architectures and/or image classification models such as YOLO (You Only Look Once) and RetinaNet.

# Appendix

For further information and step-by-step analysis and code, please refer to the following repo:

<https://github.com/sjmfirestone/comp-4449-midterm-rtfs>

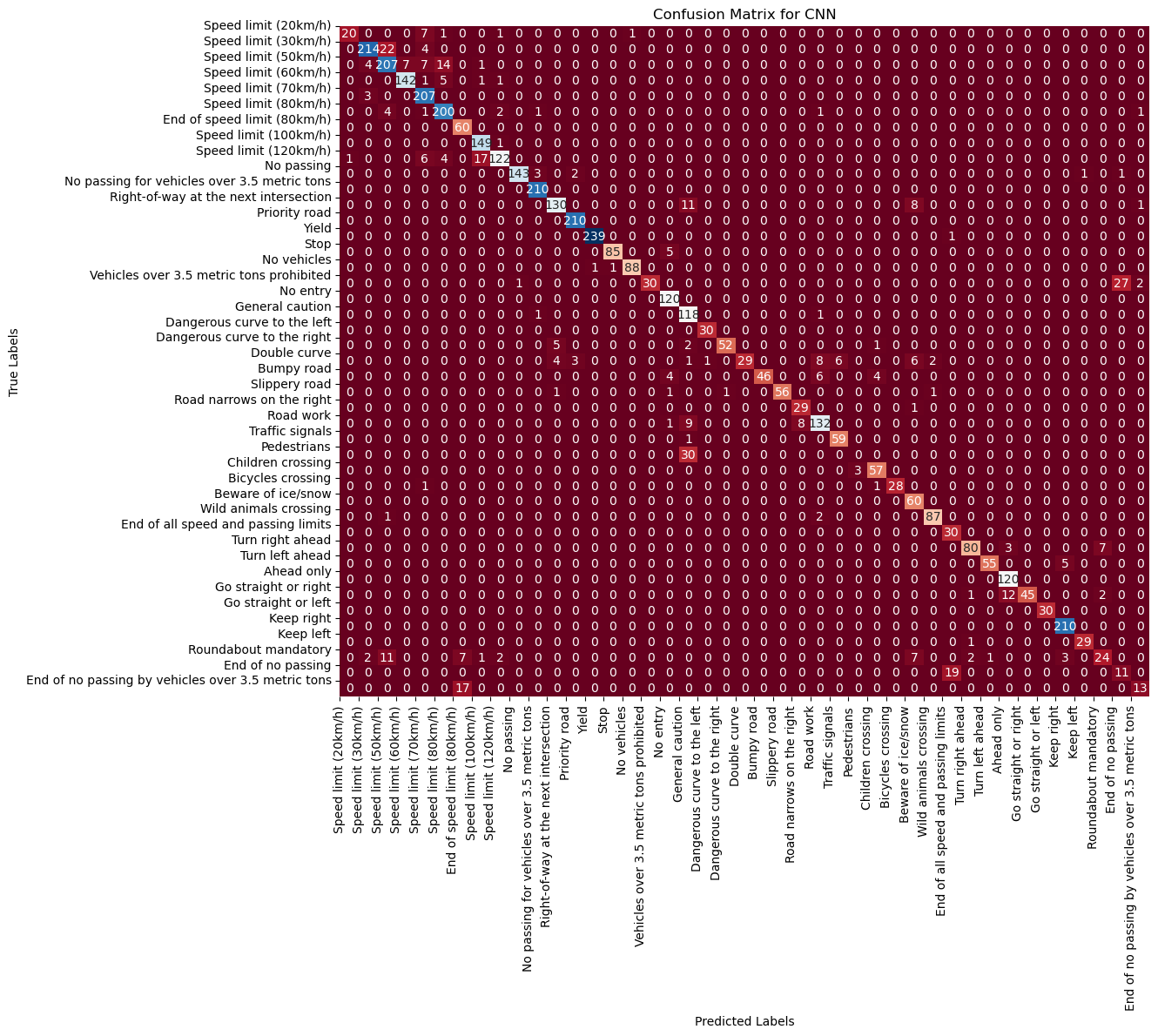


Figure 20: Basic CNN Confusion Matrix of Test Data

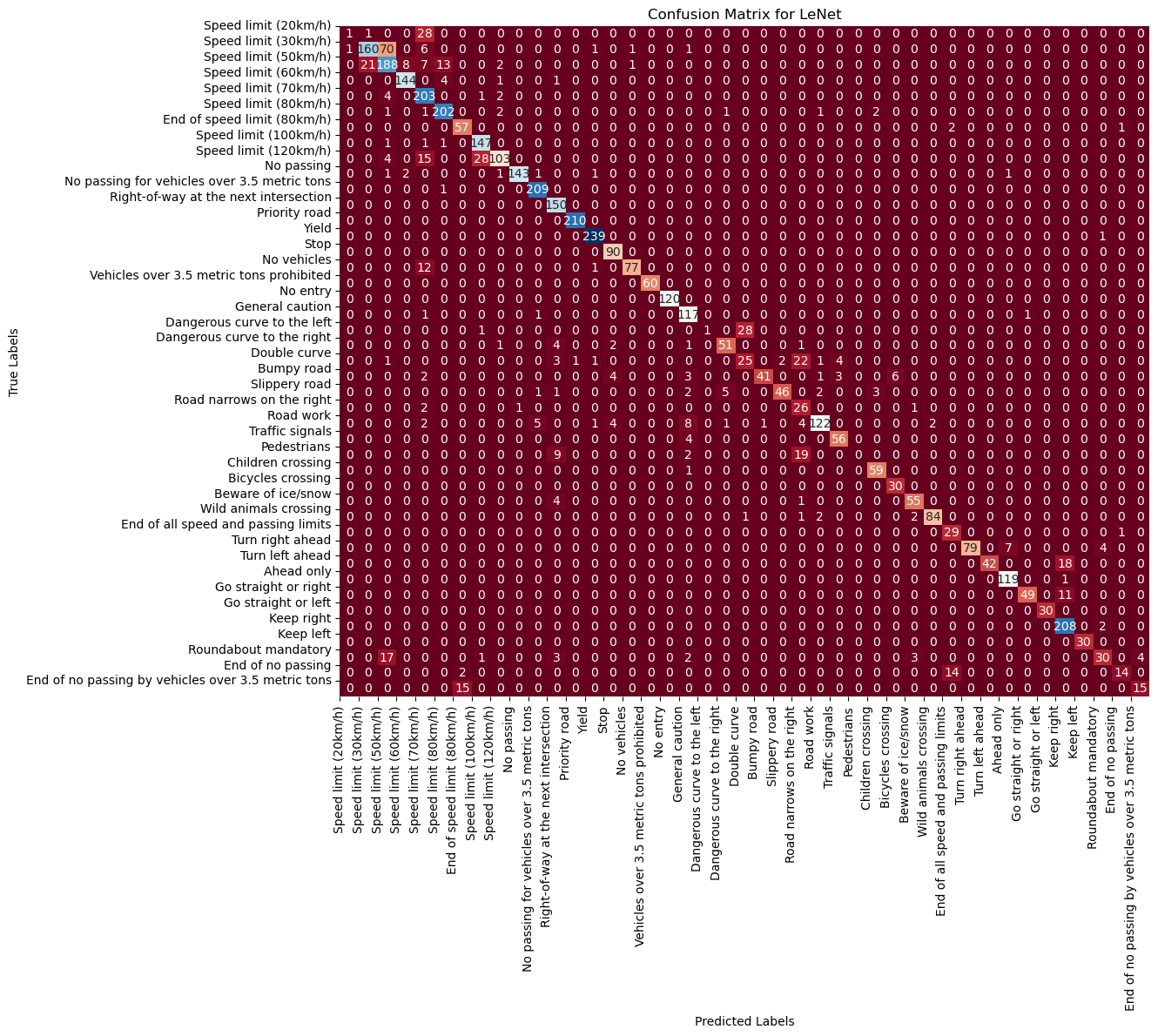


Figure 21: LeNet5 CNN Confusion Matrix of Test Data

The above confusion matrices show that the Basic CNN model and LeNet5 Based CNN model misclassify different types of traffic signs.

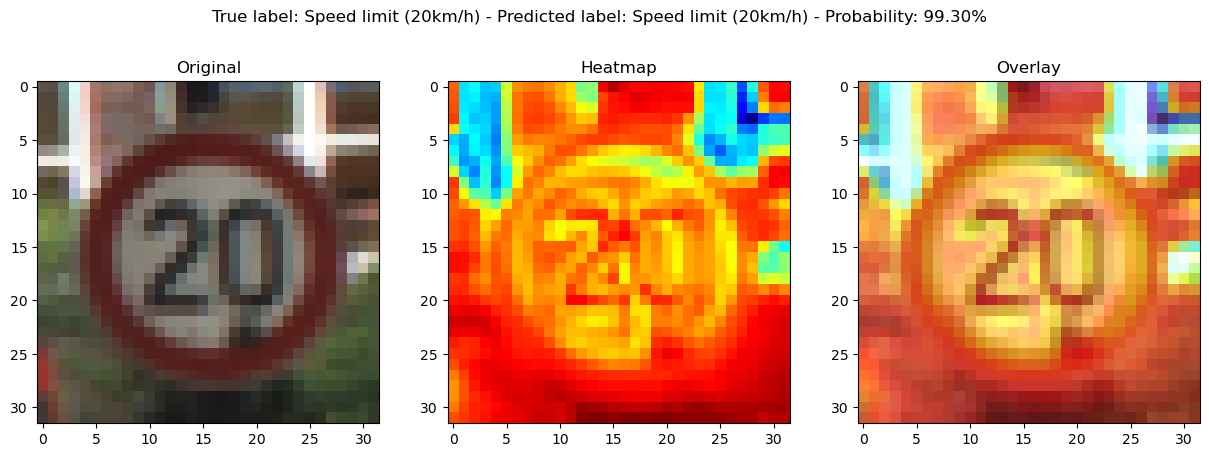


Figure 22: Basic CNN: Activation Heatmap at First Conv Layer - Correct Classification

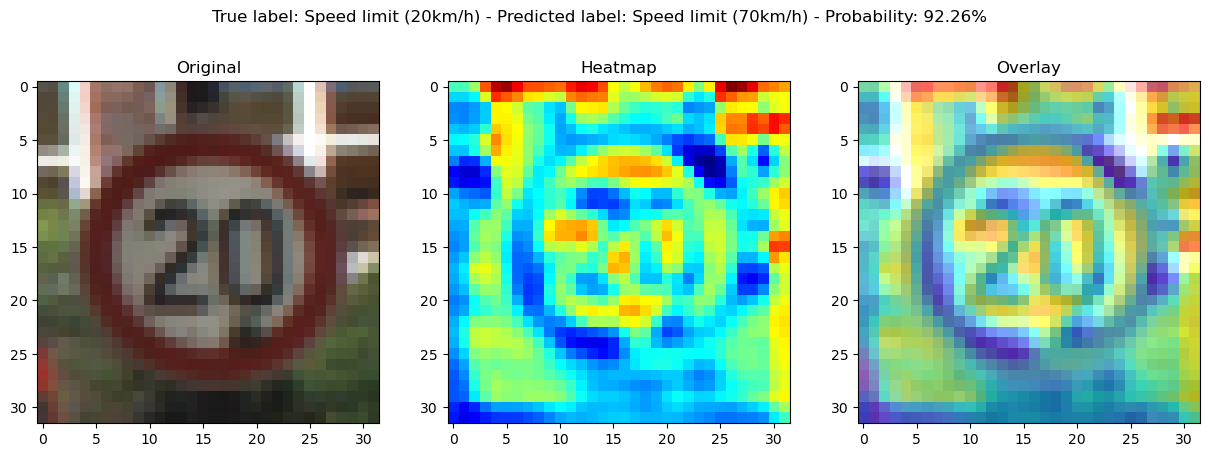


Figure 23: LeNet5 Based CNN: Activation Heatmap at First Conv Layer – Misclassification

The above activation heatmaps show that the Basic CNN model had better feature recognition on the shape of the sign as well as the numerical value `20` within the sign than the LeNet5 Based CNN model.

# References

1. Emmanuel A. (n.d.). DataScienceCapstone/3\_MidtermProjects/ProjectRTS/data. Retrieved from [GitHub repository](https://github.com/emmanueliarussi/DataScienceCapstone/tree/master/3_MidtermProjects/ProjectRTS/data)
2. German Traffic Sign Benchmarks. (n.d.). Retrieved from <https://www2.htw-dresden.de/~guhr/dist/>
3. All About Circuits. (n.d.). Image Histogram Characteristics: Machine Learning & Image Processing. Retrieved from <https://www.allaboutcircuits.com/technical-articles/image-histogram-characteristics-machine-learning-image-processing/>
4. Machine Learning Mastery. (n.d.). How to Improve Neural Network Stability and Modeling Performance with Data Scaling. Retrieved from <https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/>
5. Medium. (n.d.). Shuffle. Retrieved from <https://medium.com/100-days-of-algorithms/day-43-shuffle-b5abe4644c23>
6. Machine Learning Mastery. (n.d.). Random Oversampling and Undersampling for Imbalanced Classification. Retrieved from <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/#:~:text=Random%20oversampling%20involves%20randomly%20selecting,them%20from%20the%20training%20dataset.>
7. GeeksforGeeks. (n.d.). One-Hot Encoding of Datasets in Python. Retrieved from <https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/>
8. Saturn Cloud. (n.d.). A Comprehensive Guide to Convolutional Neural Networks: The ELI5 Way. Retrieved from <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>
9. Towards AI. (n.d.). The Architecture and Implementation of LeNet-5. Retrieved from <https://towardsai.net/p/deep-learning/the-architecture-and-implementation-of-lenet-5>
10. Analytics Vidhya. (n.d.). Complete Guide to Prevent Overfitting in Neural Networks (Part 1). Retrieved from <https://analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-1/>
11. Machine Learning Mastery. (n.d.). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Retrieved from <https://machinelearningmastery.com/gentle-introduction-to-dropout-for-regularizing-deep-neural-networks/>
12. NPTEL-NOC IITM. Explaining CNNs: Class Attribution Map Methods. Retrieved from [Explaining CNNs: Class Attribution Map Methods](https://www.youtube.com/watch?v=VmbBnSv3otc)