-Data Science Senior Project Report-
Traffic Sign Multi-class Classification Using Custom CNN

Brandon Cruz

I. Introduction

Self-driving cars have become a very popular topic in recent years due to the advancements in machine learning and computer vision. One essential component that ensures the safety of these vehicles is the ability to recognize and detect different types of objects. This project aimed to explore the recognition side of this process through the use of Convolutional Neural Networks (CNN). This project investigated the accuracy of a CNN model in classifying traffic signs, which could be integrated into a self-driving car system. The CNN model was trained using the German Traffic Sign Recognition Benchmark dataset, which includes over 50,000 images of traffic signs of different shapes, sizes, and colors. The model's performance was evaluated using two metrics, accuracy, and F1 score to assess its effectiveness in traffic sign classification.

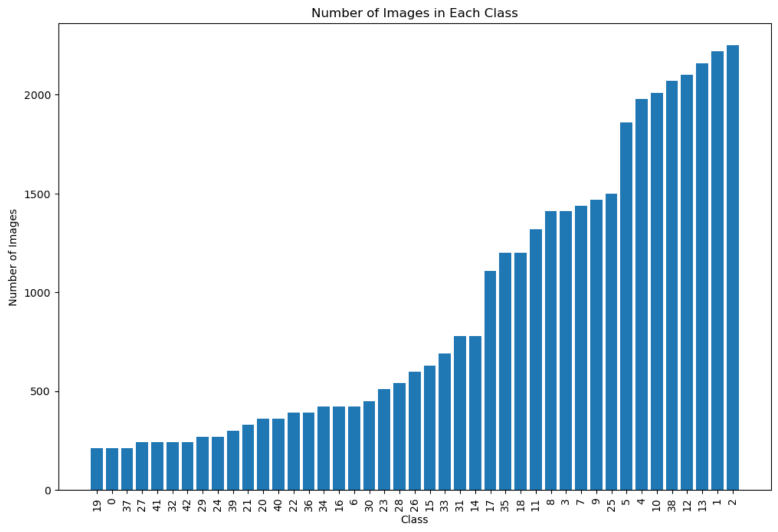
The motivation for this project arises from the lack of coverage of CNN models in my machine learning course. Therefore, one of the primary objectives of this project was to learn the process of CNN and apply it to traffic sign classification. In this project, I developed a CNN model from scratch rather than relying on pre-existing models or architectures. In doing so, I achieved a deeper understanding of the underlying concepts and processes involved in building a custom CNN model.

The remainder of this report is organized as follows. Section II discusses the data collection and preprocessing steps taken to prepare the dataset for training. Section III describes the CNN model's architecture and the different techniques employed to improve its performance. Section IV presents the evaluation results of the model. Finally, Section V concludes the report by summarizing key finds, discussing limitations, and providing recommendations for further improvements.

II. Data Collection and Preprocessing

The German Traffic Sign Recognition Benchmark (GTSRB) dataset was used as the primary source for this project. The dataset was collected by the Institute of Neuroinformatics at the University of Bochum in Germany and was made publicly available on Kaggle. The dataset was created by going through dashcam videos and cropping out the relevant portions of each video to obtain the individual traffic sign images. In total, the dataset included over 50,000 images with 43 classes.

The dataset poses some challenges for training a CNN model. One is the image size of 30x30 pixels, this small size makes it difficult for the model to capture strong details of the traffic sign and can lead to a lower accuracy. Additionally, the data distribution of images is uneven, with some traffic sign classes having over 2,000 while others having less than 500.



To prepare the dataset for training several preprocessing steps were taken. Firstly, I resized to a common size of 30x30 pixels to ensure all were exactly the same size in case some were cropped a bit bigger or smaller. To address the uneven distribution of images in the dataset, data augmentation techniques were applied to the training set to increase the number of images per class. This involved randomly flipping, zooming, and reflecting the images to create a new dataset. Once the data was augmented, I combined both the augmented dataset and the original dataset and performed a train/validation split. As a result of the split, I ended up with 62,734 images for training, 15,684 images for validation and the test data included 12,630 images. Finally, to make it easier for the model to process the data, one-hot encoding was applied to the labels. This technique transforms the categorial labels into a binary vector of length 43, where each element corresponds to a traffic sign class. This allows for better model performance and makes it easier to evaluate the model's accuracy.

Overall, the GTSRB dataset provided a diverse and challenging set of images to help train my CNN model. The preprocessing steps ensure that the dataset is clean and ready for model training.

III. Model

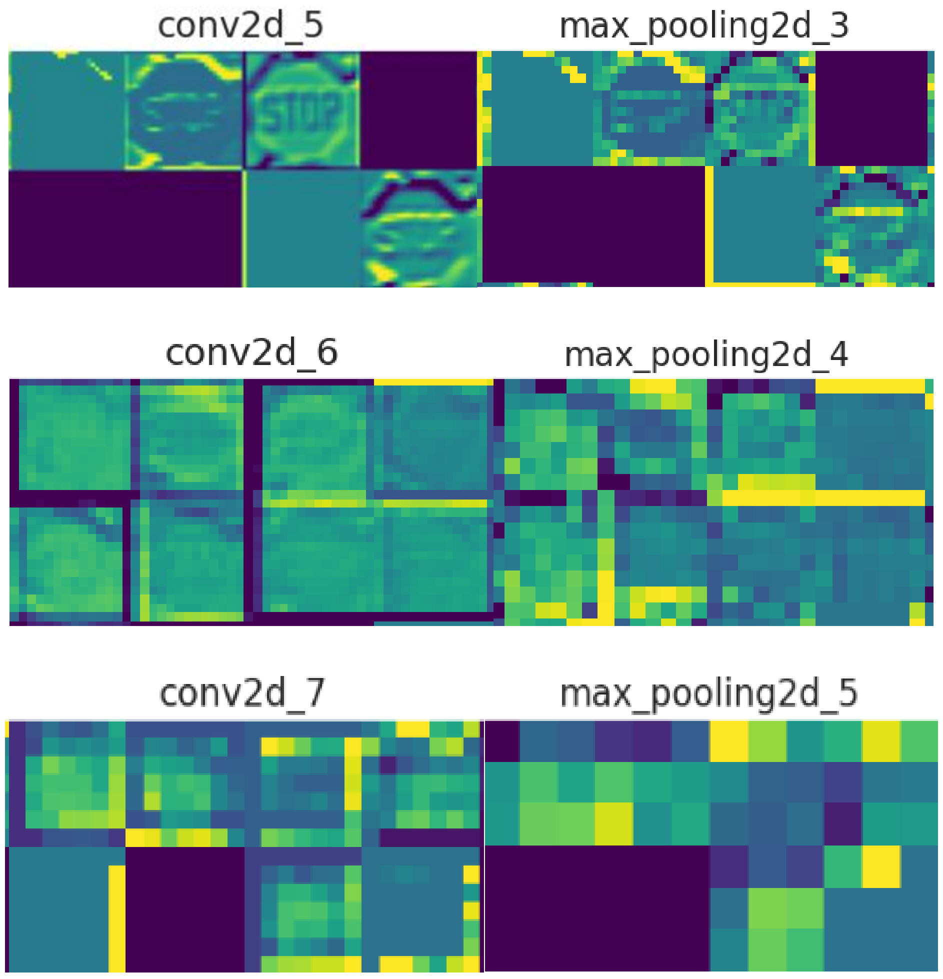
Convolutional Neural Networks are a class of deep learning neural networks commonly used for image recognition, classification, and segmentation. They are designed to recognize visual patterns directly from pixel values. The main concept behind CNNs is to extract local features from the input image by applying convolutional filters or kernels to a group of pixels. These filters slide over the image and extract the most prominent features within the filter window, resulting in a feature map. The feature map from one layer is then fed as the input image for the next layer, which then repeats the process with a different set of filters, resulting in deeper and more complex feature representations. Pooling layers are added after convolutional layers to reduce the dimensions of the output and to increase the receptive fields of the filters. Common types of pooling layers include max-pooling and average-pooling. In addition to the convolutional and pooling layers, the last convolutional layer is flattened and fed into a fully connected dense layer at the end to classify the extracted features. These dense layers use traditional artificial neural network methods to classify the features into different classes.

My model's architecture consists of four convolutional layers followed by three dense layers. I have also added batch normalization and dropout regularization to prevent overfitting. The first convolutional layer has 32 filters with a kernel size of 3x3, followed by another convolutional layer with 64 filters. The third convolutional layer has 128 filters, and the final convolution layer has 512 filters. All convolutional layers have a rectified linear unit (ReLU) activation function and use padding to preserve the spatial dimensions of the input. After each convolutional layer, I added a max-pooling layer to reduce the dimension of the output. Batch normalization layers are used after each max-pooling layer to normalize the layer’s inputs by re-centering and re-scaling, this helps improve the convergence and speed up the training process.

Dropout layers are added after each batch normalization layer to help prevent overfitting. Finally, I flattened the output from the last convolutional layers and added three dense layers with ReLU activation function. The first dense layer has 4,000 neurons, followed by a dense layer with 3,000 neurons, and another with 1,00 neurons. The last dense layer has 43 neurons, which is equal to the number of classes in the dataset. The SoftMax activation function is used for the last layer, which outputs probabilities for each class. The total number of parameters in my model is 34,129,443, of which 34,169,035 are trainable parameters and 1,408 are non-trainable parameters.

I used GridSearch to find the best combination of hyperparameters for my model, including the number of filters, the number of neurons in the dense layers, the learning rate, batch size, epochs, and seed. GridSearch is a method of systematically searching for the best combination of hyperparameters by trying all possible combinations of given variables. I ran the GridSearch for 20-22 hours and found the best hyperparameters for my model which were stated above. I ran the model using the Adam optimizer and monitored the loss function during the training. I used accuracy and F1 score to evaluate the model on the test set. Accuracy was used to measure the overall portion of correctly classified samples; however, this can be misleading. This is why alongside this metric I used an F1 score which balances both precision and recall, this takes into account both false positives and false negatives, this is better for an imbalanced dataset, typically a score above .6 is good for classification models. Ultimately, my model achieved an accuracy of 86% and an F1 Score of 0.80.

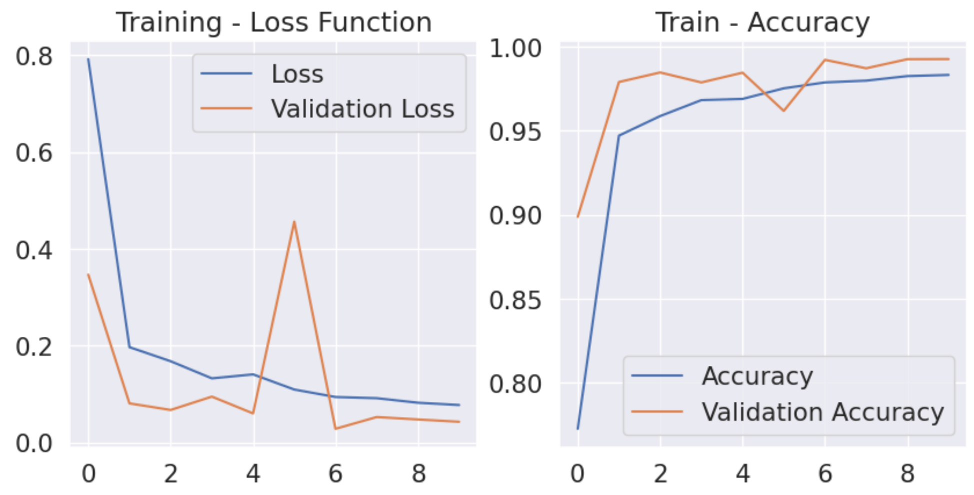
Below are examples of each convolution layer and its max-pooling layer


conv2d_5
max_pooling2d_3
conv2d_6
max_pooling2d_4
conv2d_7
max_pooling2d_5

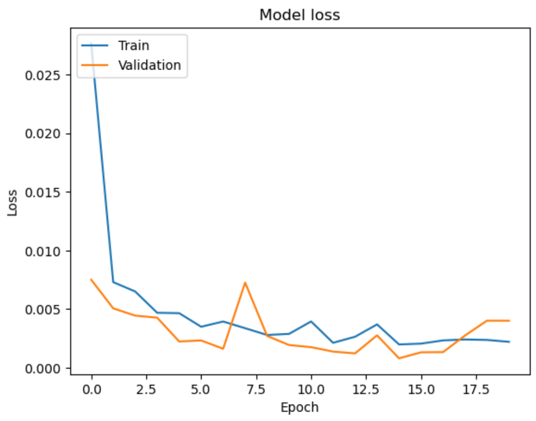
IV. Evaluation

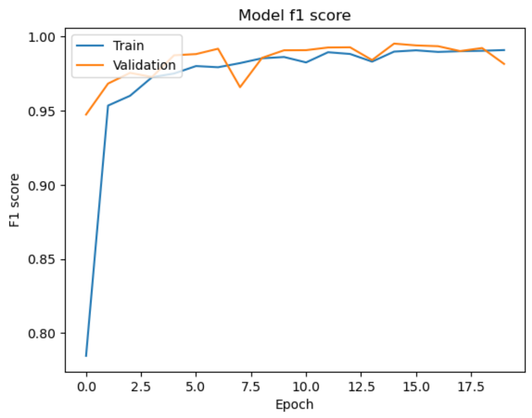
The Model achieved an accuracy of 86% and an F1 score of 0.80 on the test set, indicating that it was able to classify the traffic signs with a proficient level of accuracy. The model’s convergence was achieved within 20 epochs, which is relatively fast given the model's complexity and size of the dataset. This can be attributed to the hyperparameter tuning and the regularization techniques applied during training.

While training, I monitored the train and validation loss, accuracy, and F1 Score. In the accuracy graphs below we see accuracy start to plateau around epoch 2 and we can see the same thing in the loss. Alongside this we see a spike in loss accuracy and a drop in validation accuracy in epoch 5, this could be the model overfitting the training data, however, we see that it goes back to desired levels. One interesting observation I noticed was the fact that validation accuracy was higher than training accuracy, which could be caused by the difference in dataset splits as validation is much smaller than training, and or the regularization techniques applied to prevent overfitting.

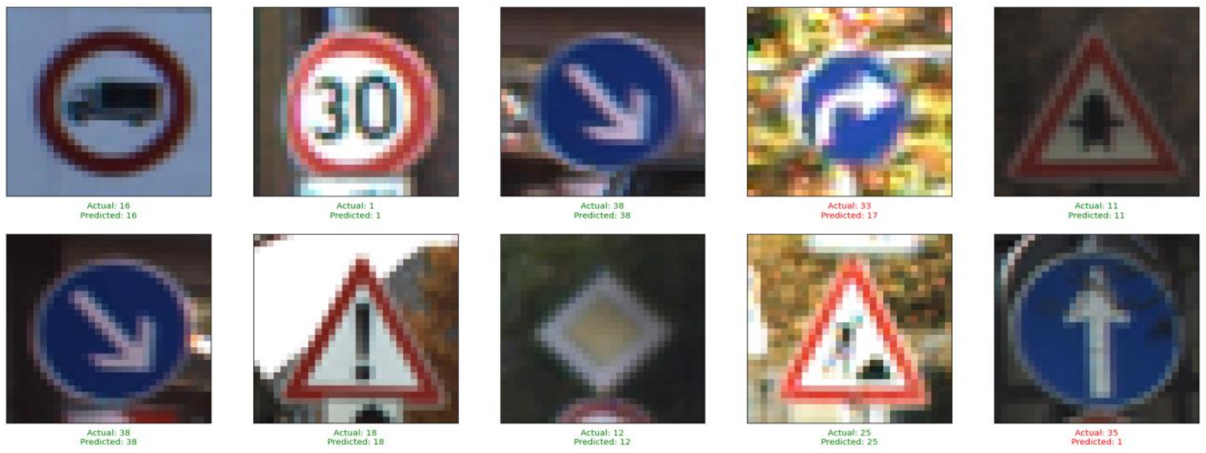


Similarly, looking at the loss graphs we can see that overall, the model did well. We see a common pattern in validation being higher than training in the f1 score graph. We see a small spike in loss around epoch 7 and a little past epoch 17.5. When comparing both accuracy and f1 score graphs we see that one has more epochs than the other. This is attributed to a call-back feature used in accuracy to stop the model once it appears to not be improving which helped the accuracy and prevent overfitting. This call-back feature is absent in the f1 score as it seemed to stop too early and by removing it, I was able to obtain a better f1 score.





Alongside graphing the model's training, I also ran a correlation matrix on the test data and found that the model had difficulty distinguishing between similar-looking signs, specifically blue circle signs with a variation of a white arrow in it. Several different classes had this variation of a white arrow, which may have caused confusion in the model in differentiating them. To see an example of my model in action I plotted a set of random signs and had my model classify them. Correct classification is in green text and incorrect is in red text.



Overall, the model performed well with a proficient level of accuracy and F1 score on the test set. However, further investigation is necessary to improve the model's performance on similar-looking signs.

V. Conclusion

Convolutional Neural Networks have proven to be a great tool for image classification tasks, including in the field of autonomous driving. In this report, I presented the development and evaluation of a custom CNN model for classifying traffic signs using the German Traffic sign Recognition Benchmark dataset. My model achieved a classification accuracy of 86% and an F1-score of .80 indicating my model does a good job at classifying images. A result I was incredibly happy with considering the dataset I was given.

However, there were limitations and challenges encountered during this project. One of the main limitations was the small input size of the images, which were only 30x30 pixels. This limited the amount of information available for the model and may have contributed to some misclassifications. Additionally, the dataset had a high imbalance class distribution, which made it difficult for the model to learn these classes effectively. The last limitation was my lack of knowledge of CNN heading into this project, as well as starting a CNN from scratch which required a significant amount of trial and error and computational power to find a suitable architecture with optimized hyperparameters.

Moving forward, there are several avenues for further exploration and improvement. A future approach is to use regional-based CNN that can identify the regions of interest in the image before classifying them. Another approach is incorporating Recurrent Neural networks to analyze temporal information from consecutive frames, which can be helpful in predicting and tracking road signs in videos. Furthermore, using a pre-existing model could help greatly improve performance.

My work highlights the potential of CNNs for road sign classification in developing recognition systems for self-driving cars. The ability to accurately detect, recognize and classify a road sign is a critical part of keeping these cars safe and efficient, and my model shows how effective CNN when tackling these problems.