Road Sign Recognition [Project Writeup]

Marvin Larweh, Maximus Berner, Qing Yan, Yuchen (Charles) Li

1 The Problem

We aimed to develop a Traffic Sign Recognition (TSR) employing a Convolutional Neural Network (CNN), with the overall goal to identify and distinguish traffic signs accurately. The program will take a still image containing a road sign (in any orientation) as input and produce a classification as output.

2 Importance

Quickly abstracting information from traffic signs is an essential task for safe and lawful driving. Most traffic signs provide crucially relevant information that often necessitates an immediate response from the driver, making it an key component of and prerequisite to autonomous vehicles. TSR systems are extensively researched and integrated within autonomous vehicles to help them avoid dangerous situations, reduce accidents, and ensure the safety of passengers and other road users. With that said, the task poses challenges due to variations in angles of perception, lighting conditions, and the diversity of signs. Additionally, many road signs are set up temporarily to respond to an immediate threat to safety, emphasizing the importance of versatile and accurate recognition programs. We take for granted being easily able to assess the varying different contexts and environments we drive in – something that autonomous vehicles will need to perfect in order to share the road with human drivers.

3 Data Being Used

There are many compiled databases of road signs with open access. Among these, we chose to use the German Traffic Sign Recognition Benchmark (https://sid.erda.dk/public/archives/daa
archive.html), particularly because of its extensive database of labelled training data and history of reproducibility.

4 Model and Workspace Being Used

We used Convolutional Neural Network due to its high effectiveness in image processing and learning hierarchies of features. The CNN architecture would also be a practical choice for real-world applications of this algorithm due its fast performance on embedded systems. For our workspace, we decided to use python for its extensive choices of libraries in building CNNs. From TensorFlow, we used Keras to build, compile, and train the CNN model. Datasets, layers, and models were used to simplify the process of building predefined layers (including Conv2D, MaxPooling2D, Flatten, Dense, and Dropout), tweak hyperparameters, and load in our dataset. ImageDataGenerator allowed us to augment our training data through transformations including rotations, shifts, and flips – each of which improved the generalization of our model. We also were able to use matplotlib.pyplot to visualize our training/validation accuracy and loss, allowing for better analysis.

5 Results

Our original CNN model is represented in Figure 1, utilizing a series of two 2D Convolution (Conv2D) layers followed by a MaxPooling 2D layer. The first pair of convolutional layers extracted low-level features (i.e., basic patterns) from the input images, and the second pair extracted more complex, high-level features. The MaxPooling2D layers that followed worked to decrease computational overload, abstract general features, and reduce overfitting. At the end of the sequence, the data was flattened into a 1D vector, condensed to 256 neurons using a Dense layer, was randomly reduced by a factor of 0.5 by a Dropout layer, and finally reduced into 43 neurons to serve as classification. This model performed with a testing accuracy of about 74%. We implemented three main

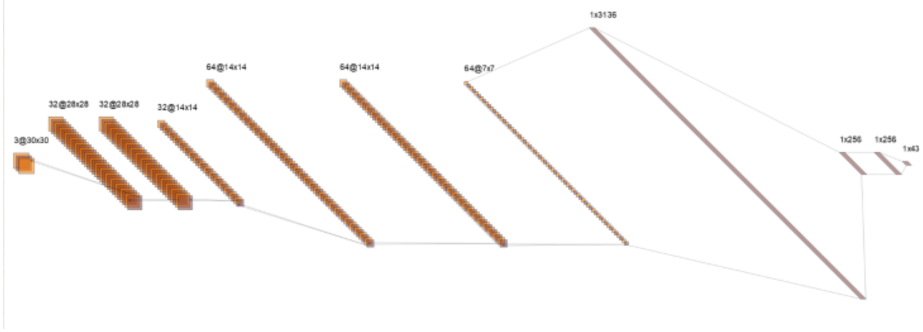

Figure 1: Visualization of first implementation of our CNN

changes to our sequential CNN to improve this. First, we added two additional Conv2D layers and one additional MaxPooling2D layer before the flattening and condensing layers. This helped to extract more features from the training images and improve the model's overall performance. Secondly, we increased the Dense layer at the end of the network to work on 512 units as opposed to the original 256. Having a higher density in this layer allowed our model to learn high-level representations of the data. Finally, we reduced the Dropout ratio from 0.5 to 0.05. While including a Dropout layer prevents potential overfitting of our model, reducing the ratio of neurons that were randomly dropped allowed the model to retain as much feature information as possible to make classifications. After these alterations were made, our optimized model achieved a testing accuracy of 88.2%. This demonstrates a marked improvement of our optimized model over the original model.

6 Difficulties Encountered

When constructing a CNN structure, there are many hyperparameters that need to be considered for a reasonable performance, such as number of layers, filter sizes and so on. Exploring all possible combinations is computationally infeasible, especially as the number of hyperparameters increases. In addition, hyperparameters often interact in nonlinear ways with each other, meaning increasing some might require adjustments to the others to maintain the performance. It is usual when tuning one parameter in isolation leads to suboptimal results because of such dependency. Another problem is the risk of overfitting in our case. The high accuracy in the training data set is sometimes misleading when the model is applied on the test data set. Too many parameters can lead to the model memorizing the training data instead of generalizing to unseen data. Furthermore, training CNN is computationally expensive for large datasets considering that we have over 50,000 images and different parameter configurations. Even a small change in some parameter or network layout requires training from the beginning, which consumes a lot of time and effort.

7 Implications and Future Work

Many supporters of autonomous vehicles often argue that the essential threshold for them to achieve is matching the performance of an average human. As the argument unfolds, if the autonomous driver can match or exceed the capabilities of the average human, their deployment would result in a reduction in accidents. This has real world consequences, with 42,514 deaths in the United States in 2022 being attributed to motor vehicle accidents (https://www.iihs.org/topics/fatality-statistics/detail/state-by-state). Autonomous vehicles are arguably approaching the capabilities of humans when it comes to driving, but it is known that these vehicles operate best within predictable and ideal conditions. Humans are comparatively better at assessing unfamiliar situations – and unexpected conditions can be the trigger for fatal accidents. Road signs can be crucial to conveying unexpected conditions, whether it is a temporary obstacle in the road or providing context of unsafe features in a particular area. With this in mind, traffic sign recognition is an essential aspect of safe driving, and the acceptable margin of error is quite thin. With our CNN, we were able to consistently achieve testing accuracies of mid 80%. This is a respectable number, especially considering the dataset we are using has extremely difficult images to distinguish even with the human eye – and that is before any augmentations like rotating/flipping/etc. However, even with this accomplishment, it is clear that with lives at stake a higher accuracy should be sought for. Some estimates place the average number of road signs we encounter at 10 signs per mile (https://signworksthinks.com/how-many-road-signs-are-there-in-the-us/), and even with 85% accuracy, it is highly unlikely that our CNN could accurately identify all signs it encounters in only 2 miles of driving. While we have respectable accuracy for classifier with over 30 different classifications, this shows that this particular classification needs to meet extremely high standards. Ultimately, improving upon our model requires a more extensive dataset, much larger time allocated to training, and hopefully better fine-tuned hyperparameters – all of which can be done building off of our model.