Recognizing Traffic Signs

Thad Hoskins

Autumn 2021

1 Introduction

"The goal of this project is to build a model capable of determining the type of traffic sign that is displayed in an image captured under different real-life conditions and showing obstructions, poor lighting, or even the sign being far away from the camera."

Image recognition and Neural Networks have many general applications. However, for this project, the goal of correctly identifying traffic signs has a very "real world" application: autonomous driving cars.

There are many on-going projects and even competitions to optimize this to near 100% accuracy. My goal at the outset was greater than 95% accuracy. For the real-world application, a high threshold of accuracy is necessary. Multiple pictures of a sign may be taken in a split second to gain the confidence the system needs to know the sign with certainty. A 1% failure rate could be mistaking a stop sign for a yield sign or missing a speed limit.

2 Data Analysis

The data for images for model training, validation, and testing were provided in example code, so data procurement was a simple matter of running the code.

- Training set – 34799
- Validation set –12630
- Test set – 4410

2.1 Inspection of the Signs

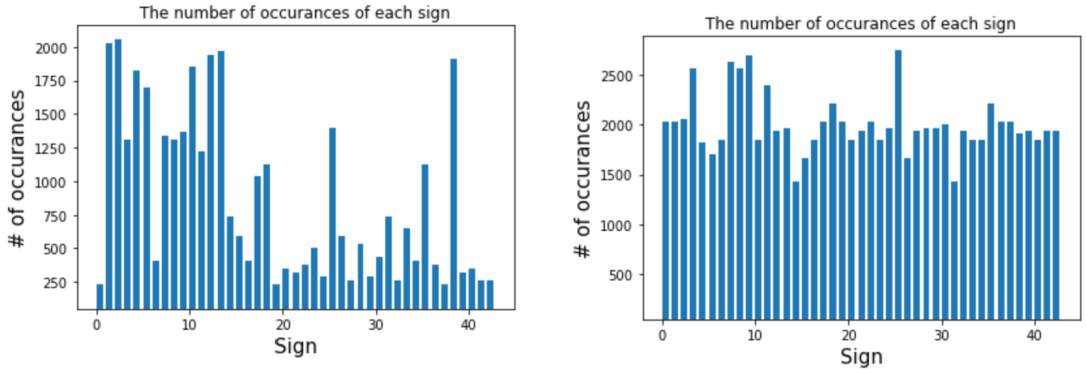
The signs are in 43 categories or types of signs, e.g., "Speed limit (50km/h)" or "Road narrows on the right". To begin, I "graphed" a sampling of the signs. Some of the signs are clear and unmistakable but some are dark enough that I cannot tell what they are. Each image is 32x32 and they are in color.

2.2 Data Distribution

Using a graph of the distribution of the signs, we can see that the data is not evenly distributed. This could be a problem as certain signs would be "seen" much more often than other signs. Were this a person, they become more familiar with certain signs and not others. Were they then to take a guess, it could be they would guess what they see more often. Computers may not think exactly this way, but the point is that the learning may not be as good without more evenly distributing the data.

Figure 1: Sampling of Signs

Figure 2: Sign Distribution Original vs Equalized



Using augmented images already in the dataset, rotating the images and changing the brightness/contrast, I filled in additional images to equalize the counts. Many Python libraries were available to do this. One such example is: https://github.com/vxy10/ImageAugmentation

The solution equalized the distribution as the graph shows.

However, testing indicates the accuracy was not improved. Given the cost of the equalization computation, the significant increase in images for training, and the lack of improvement, the method was dropped.

2.3 Image Augmentation

There were many methods and techniques for augmenting the data. A few are listed below:

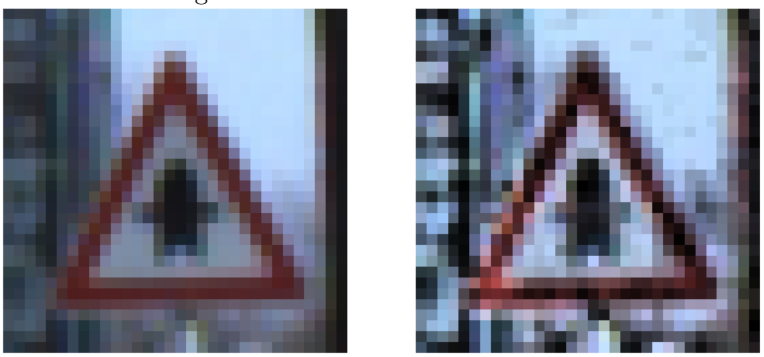
- Convert to black and white
- Brighten the images

- Normalize the values of the pixels
- Rescaling

The "winning" techniques mentioned above (greater than 99%) do all of the above, especially converting to black and white. However, their techniques of doing so are unknown to me. When I used common techniques for those outcomes my results were worse than leaving the images in their original form.

Normalizing was a favorite and I was hopeful for that one as it increases contrast. However, this technique appears to help a human but not a computer.

Figure 3: Normal versus Normalized



In the end, I left the images in their original form.

2.4 The Model

2.4.1 Convolutional Neural Network (CNN)

Neural Networks generally operate using an input layer, hidden layer(s), and an output layer. There are different kinds of Neural Networks that have specialize purposes. CNN are generally used for classification. In this case, we will use it for image classificaton.

"a mathematical operation on two functions that produces a third function expressing how the shape of one is modified by the other."(Wikipedia)

CNN are made of hidden layers (Convolutional and Pooling layers), and Fully-connected layer leading to output.

The main computation and magic happens in the Convolutional layer. The sections of this layer are data, filter, and feature map. Our image is a 2 dimensional image with a color, therefore the input has 3 dimensions, width, height, and depth (RGB color channel). The filter is the feature detector. This is the main classification piece, looking for features that can identify differences and similarities of images.

The pooling layer is similar but the output is the pixel that has the greatest impact. Ultimately, this layer's job is to reduce complexity. Data is lost but, importantly and hopefully, the most important information is retained.

The two previous layers use ReLU (Rectified Linear Unit) for their activation functions which adds a non-linear component to the model.

The full-connected layer is the last layer, connecting the previous layers with the output layer. The task of classification is done in the fully-connected layer. In the case of my CNN, the fully-connected layer uses softmax to output probabilities that an image is one of the 43 output classifications.

For this project, a probability of 0.06078681 for any single classification vector may be interpreted as 100% or highest confidence.

2.4.2 Tuning

For the tuning I used a few techniques, all with similar outcomes. They are manually-automated versus fully automated techniques. Keras has a tuning library that uses Hypermodelling, where one builds the model giving ranges to chosen parameters, including step counts, minimum, and maximums. The parameters are tuned using RandomSearch. In the end, this did not improve the model noticeably over other tuning methods. This method also retains in the file system previous tunings. This was very helpful to maintain state.

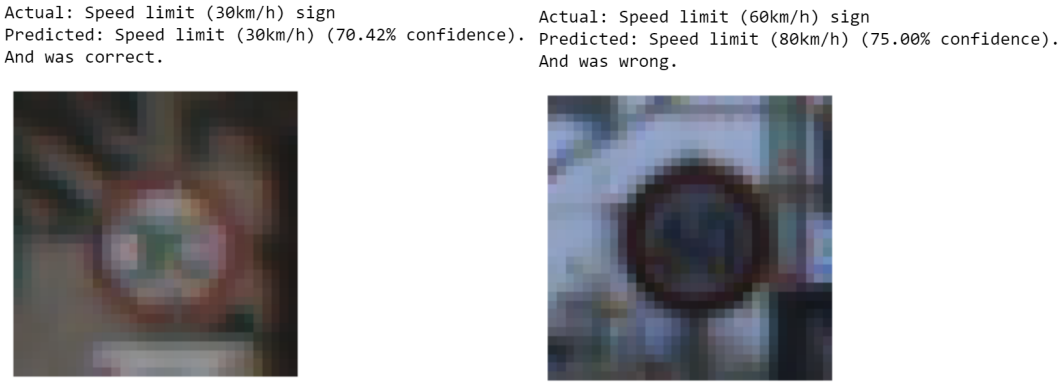
The other method was to manually create a function that creates the model then use GridSearchCV to tune the hyperparameters. In this case the main tuning is the optimizers.

Both have their pros and cons. Further use of both may yield better results rather than turning dials in the dark. For the experience and with a desire to tune the specific hyperparameters of the CNN, I chose the Keras tuner library with a RandomSearch.

2.4.3 Results

The final results are positive. The model could classify the signs well. It identified multiple signs in which I could not tell what the sign is. However, certain image artifacts would confuse the model and it could not accurately classify the sign, low lighting being the most deterministic. The final validation accuracy was 0.9859410226345062, with a test accuracy of 0.9762470126152039. Given further, more sophisticated tuning the accuracy could improve but I was very happy with the result.

Figure 4: Correct and Wrong Classifications



Depending on the source, a human is between 98.4 and 99.2% accurate with the same dataset. This means that while the computer may miss that 1%, so might a human driver. With more tuning, the computer can noticeably outperfom a human.