Traffic Sign Classification Report

1. Goal /Steps of this project

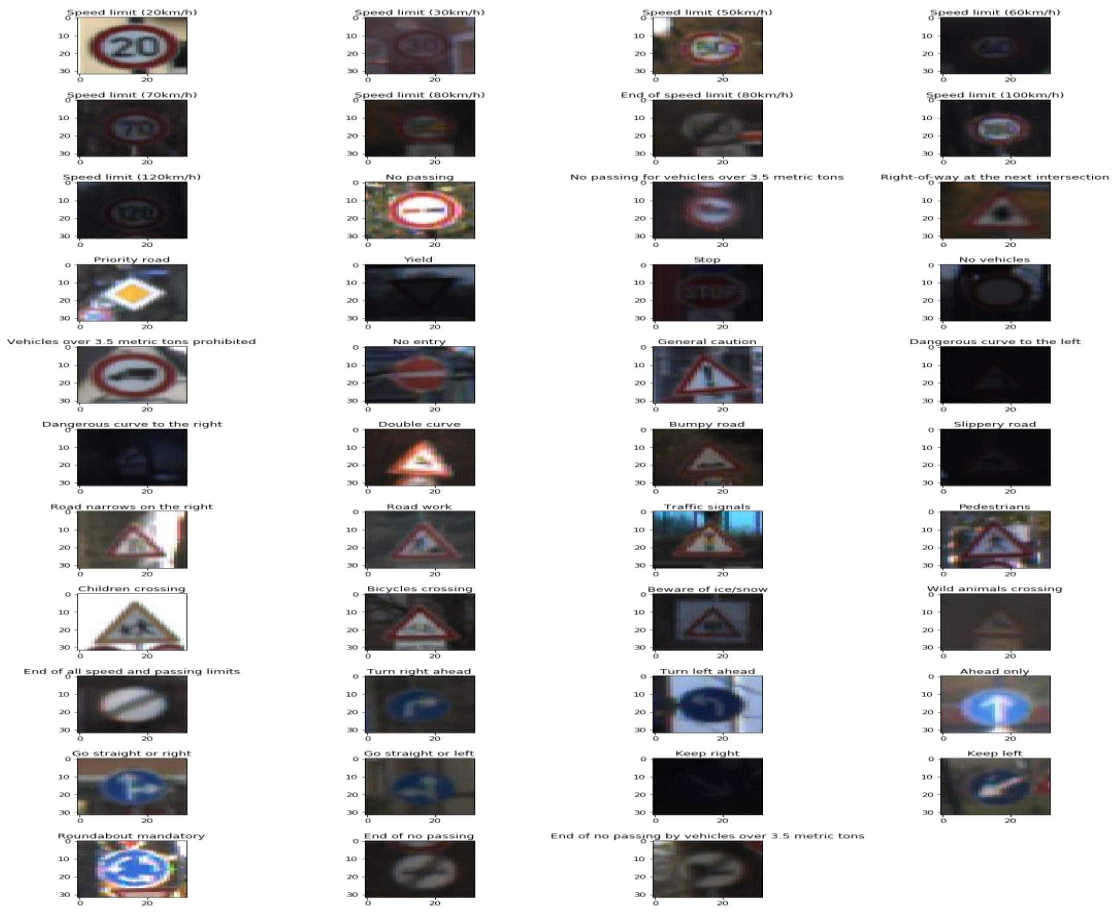
- Load the data set
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

2. Dataset summary

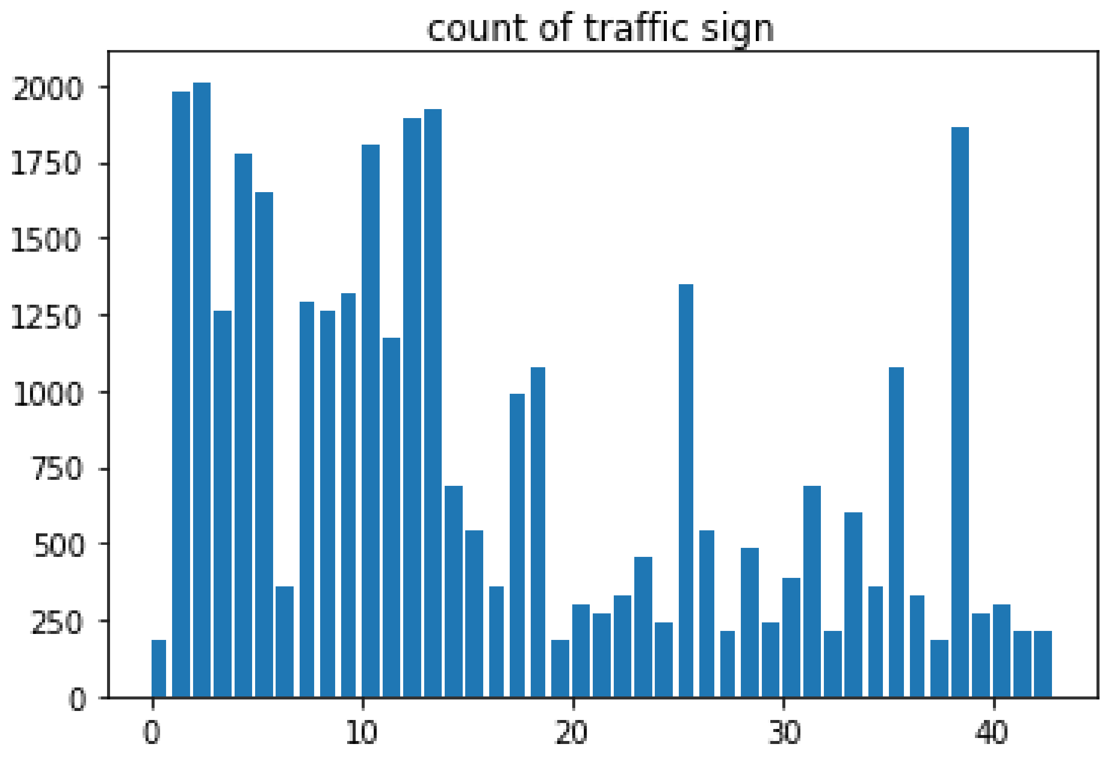
Here is the summary of the size and shape of the dataset I am using for training, validation and testing.

- Number of training examples = 34799
- Number of testing examples = 12630
- Image data shape = (32, 32, 3)
- Number of classes = 43

The picture below shows the 43 classes in the German traffic sign data set, along with the same of each unique sign. (For a clearer version, please refer to my notebook source code)



The following histogram picture shows the total number of each traffic sign. As you can see some of the traffic signs have a bigger number of other traffic signs. This might leads to a better prediction for those traffic signs with more samples.



3. Model Architecture

3.1 Preprocessing

Firstly, I shuffled the training data set, because I don't want the order of the data be a factor of my training accuracy.

For the preprocessing part, I applied two basic preprocessing of the dataset, Grayscale and normalized. Color information can be helpful for classification. But for the traffic sign classification, the shape of the sign is a more important indication for different signs. Also, in different environment condition, the color of the same sign might also differ, which is likely to cause some confusion. That's why I decided to grayscale the dataset and meanwhile also I manage to change the depth of the data set to 1, which fits my training model.

Besides, I normalized the data into range [-1, 1], because a dataset with mean zero and equal variance leads to a better accuracy.

3.2 Model architecture

I use the **LeNet-5** implementation from the structure as my starting point of my deep learning model. However, it couldn't reach a validation accuracy of more than 0.93.

Then I tried to figure out what could be the reason of it. I tried to use different way of normalization when processing the data, and I also tried to tune the learning rate and the batch size. But none of this trial was successful.

Then I started to keep tracking of both the accuracy of the training dataset and the validation dataset. I found that the problem of the low accuracy is actually caused by overfitting. Therefore, I decided to adding Dropout in the Lenet5 model to overcome the overfitting issue and it finally worked.

Please refer to the following table for the detailed implementation of my deep learning model.

Layer	**Description**
Input	32x32x1 Grayscale image
Convolutional layer	Input 32x32x1. Output 28x28x6 strides=[1, 1, 1, 1], padding='VALID'
Activation	Relu

Max pooling	Input = 28x28x6. Output = 14x14x6
	strides=[1, 2, 2, 1], padding='VALID'

Convolutional layer	Input = 14x14x6. Output = 10x10x6
	strides=[1, 1, 1, 1], padding='VALID'

Activation	Relu

Max pooling	Input = 10x10x16. Output = 5x5x16
	strides=[1, 2, 2, 1], padding='VALID'

Flattening	Input = 5x5x16. Output = 400
Fully-connected layer	Input = 400. Output = 120.
Activation	Relu
Dropout	Keep_probability = 0.5
Fully-connected layer	Input = 120. Output = 84.
Activation	Relu
Dropout	Keep_probability = 0.5
Fully-connected layer	Input = 84. Output = 43

3.3 Model training

Please refer to the following table for all the hyperparameters I use for training.

Epoch	**20**
Batch Size	**128**
Learning rate	**0.002**

The optimizer I use is the Adam optimizer since it is empirically proved to be a powerful optimizer that can achieve good results fast even with noisy/or sparse gradients.

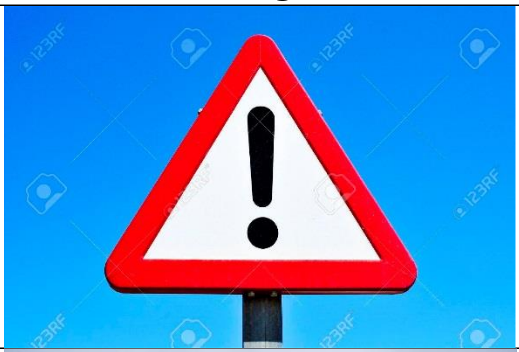
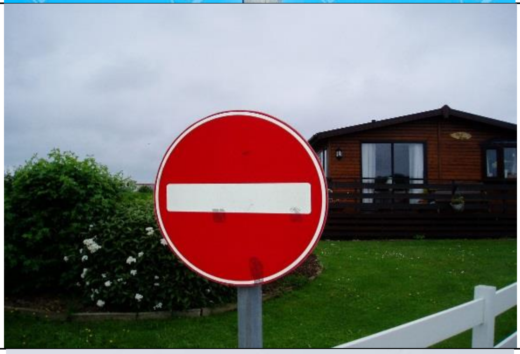
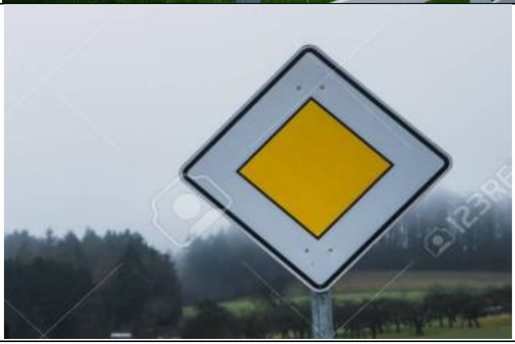
3.4 Final result of my model

- Validation Accuracy = 0.951
- Training Accuracy = 0.994
- Test Accuracy = 0.931

The validation accuracy with a value 0.951 is bigger than the minimum requirement of 0.93.

4. Test a model on New Images

The table below shows the image I used and the prediction results.

Image	Prediction
	General caution: 100.0% Speed limit (20km/h): 0.0% Speed limit (30km/h): 0.0% Speed limit (50km/h): 0.0% Speed limit (60km/h): 0.0% Prediction is right
	No entry: 100.0% Speed limit (20km/h): 0.0% Speed limit (30km/h): 0.0% Speed limit (50km/h): 0.0% Speed limit (60km/h): 0.0% Prediction is right
	Speed limit (30km/h): 28.1% Speed limit (50km/h): 26.0% Roundabout mandatory: 15.1% Priority road: 6.1% Speed limit (80km/h): 3.9% Prediction is wrong

	Speed limit (20km/h): 43.8% General caution: 13.5% End of speed limit (80km/h): 13.0% Speed limit (120km/h): 10.2% No entry: 4.1% Prediction is right
	Speed limit (60km/h): 100.0% Speed limit (20km/h): 0.0% Speed limit (30km/h): 0.0% Speed limit (50km/h): 0.0% Speed limit (70km/h): 0.0% Prediction is right

The reason of choosing these five pictures are as follows:

The first one is a very easy and straightforward one with sharp edge with the sign in the middle of the picture. It is used as the bottom line to check that my model works.

The second one is still quite clear but the picture is darker and the size of the sign is also smaller, which might be a bit of a challenge to classify.

In the third one, the sign has a big white part, which is very similar to the background of the picture, which might increase the difficulty.

The fourth and the fifth ones are quite similar signs with different number. I used these two pictures to check if my model can tell the different number of the sign correctly.

The result shows that my model correctly classifies 4 out of the five images. It failed at image 3. From my perspective, the reason is because when it is converted to grayscale, the yellow and the white edge is getting blurred. This will make the prediction become harder.

For image 1, 2 and 5, the prediction is very accurate with only the right label with 100% probability.

However, for image 4, the prediction is also quite interesting. Although the prediction is right, some other traffic signs that are very similar to this one also have quite high probabilities.

Overall, I think my model did a good job of identifying 4 out of the 5 images correctly.

5. Further Improvement

Even though the final result passed the requirement of this project, there is still room for improvement. I have already had some ideas that might help.

- Augmentation.
  As we can see from the histogram chart, some traffic signs have enough data, but some only have a few samples. Augmenting the dataset, especially for those with a small number of samples, can improve the accuracy.
- Preprocessing
  Now I only applied grayscale and normalization. But when it comes to real environment, it requires more preprocessing. For example, color space translation.