Build a Traffic Sign Recognition Classifier
Udacity SDCND Project 2

6/21/2017

Daniel Tobias

1. Dataset Summary and Exploration

The goal of this project was to train and evaluate a convolution neural network to detect and classify traffic signs. To simplify things a dataset of german traffic signs were used since they were labeled and the images were standardized across the dataset. The dataset had 43 different types of traffic signs visualized below.

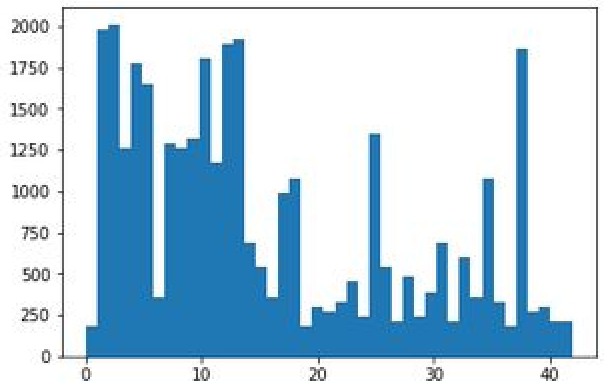
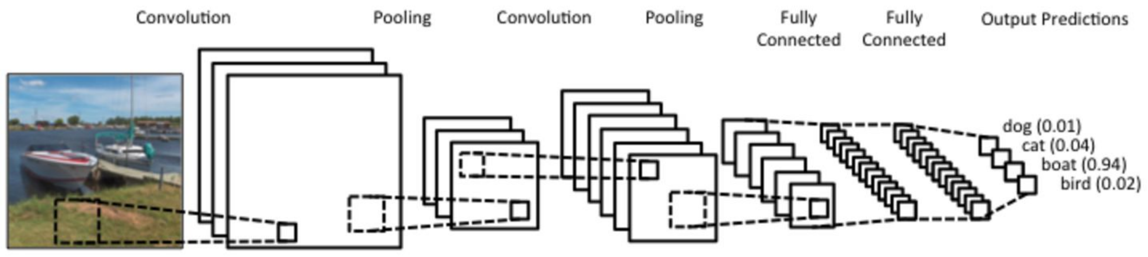


Figure 1

From the histogram it is clear that there is a large difference in labeled images for each traffic sign type. This could be problematic later in training, for instance, the error could be one entire traffic sign that could not be correctly generalized to. To counter the large inequality in samples I decided to shuffle the images. This helped to diversify the mini batches with more types of traffic signs.

2. Design and Teach a Model Architecture

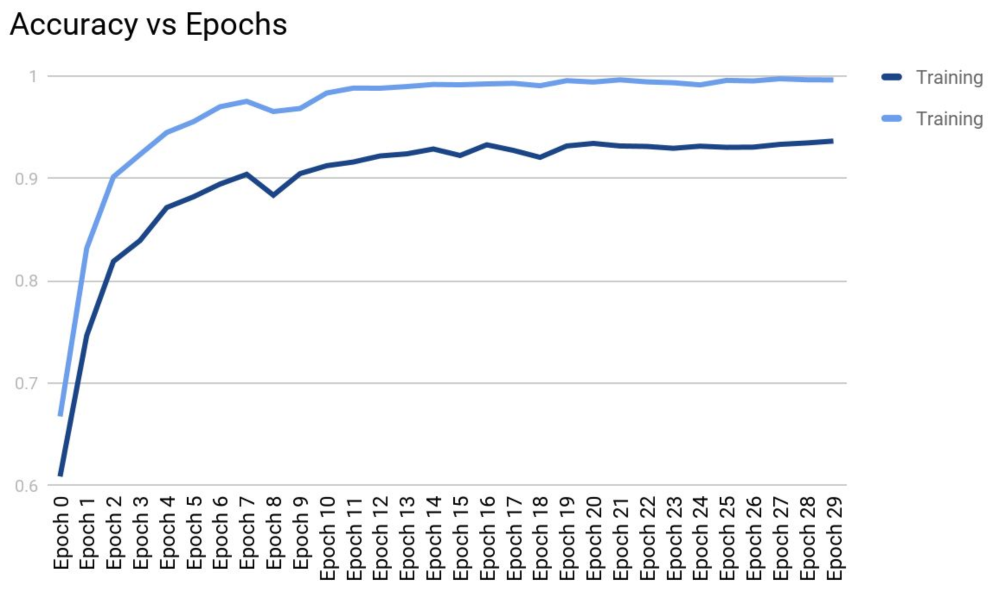
The two things that were done to preprocess the images were shuffling that data and turning the images to grayscale. I picked grayscale first and the results were good enough that I decided to keep it instead of using color images. I fed the images through a neural net architecture very similar to LeNet. The image below characterizes and describes the architecture I used.



https://ireneli.eu/2016/03/13/tensorflow-04-implement-a-lenet-5-like-nn-to-classify-notmnist-images/

Operation	Dim In	Dim Out
Grayscale	32x32x3	32x32x1
Conv1	32x32x1	28x28x6
Max pool1	28x28x6	14x14x6
Conv2	14x14x6	10x10x16
Max pool2	10x10x16	5x5x16
Flatten	5x5x15	400
Fully Con1	400	120
Fully Con2	120	84
Output	84	43

The stride and pooling parameters and etc, are basically the same from LeNet architecture for classifying the MNIST dataset. The videos from the class about the LeNet architecture is where I started from.A learning rate of 0.001 using the adam optimiser which was used along with a batch size of 100 for 30 epochs. Luckily I have access to a GTX 1080 which made training relatively fast.The final validation accuracy I got as 93.65%, and the validation set was roughly 8.5% of the dataset used. The final training accuracy was 99.6%. The resulting accuracy vs epoch is detailed in the chart.



3. Test a Model on New Images

The final part of the project was to evaluate the training model on other traffic signs collected from the internet.

Stop Sign, ID #14

Right of way next intersection, ID #11

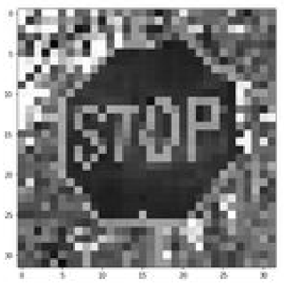
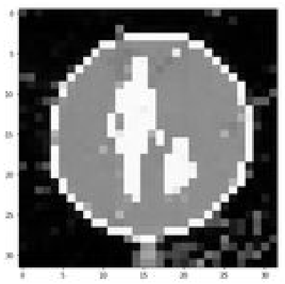
Roundabout, ID #40

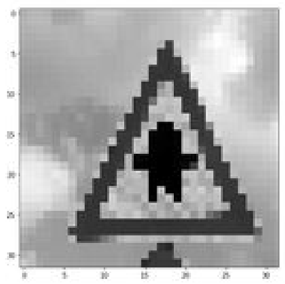
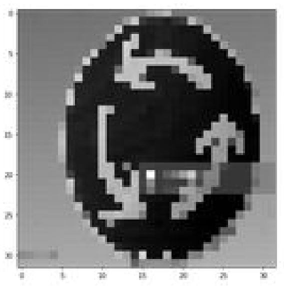
Children Crossing, ID #28

Speed Limit, ID #1

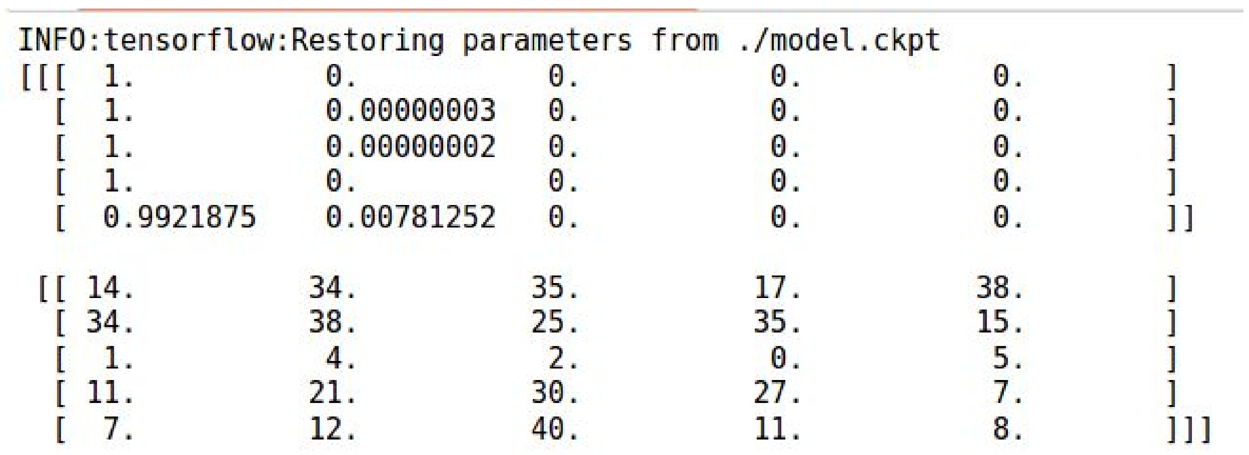
The above images were used to to test out the system, I deliberately chose german traffic signs to test the accuracy of the trained model.

The same images were resized and turned to grayscale to match the input data used to train the model.

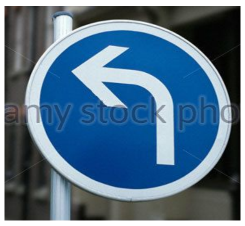



Using the model, it correctly predicted 3 out of 5 traffic signs, so an effective accuracy of 60%.



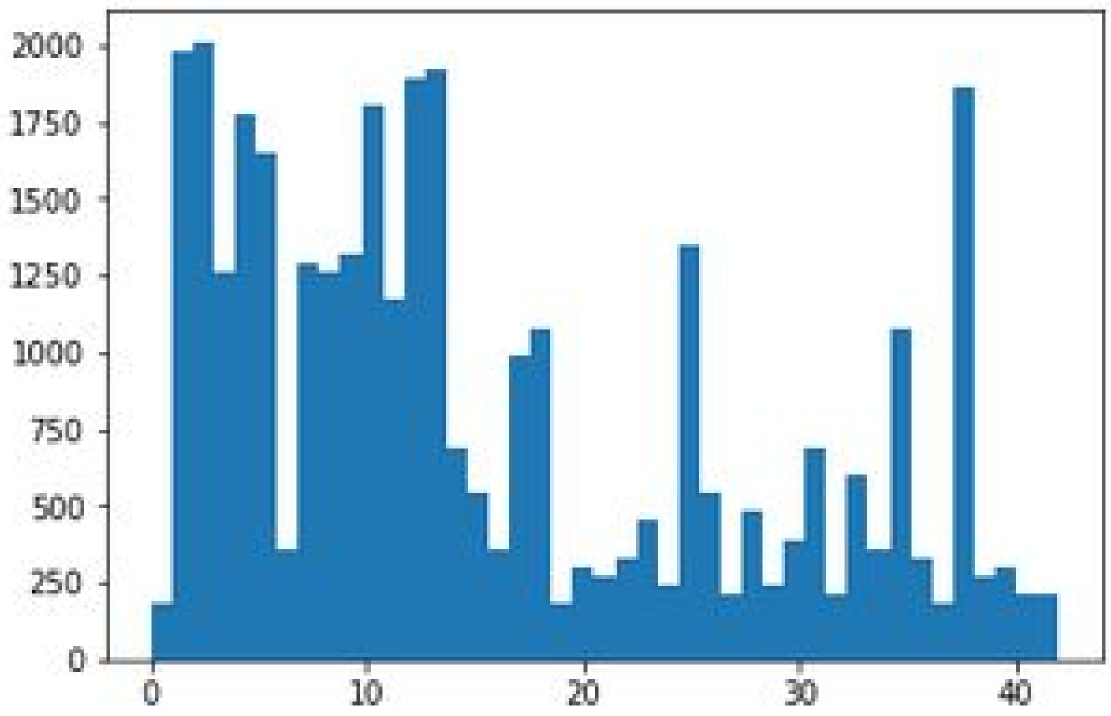
```
INFO:tensorflow:Restoring parameters from ./model.ckpt
[[[  1.          0.          0.          0.          0.        ]
  [  1.          0.00000003  0.          0.          0.        ]
  [  1.          0.00000002  0.          0.          0.        ]
  [  1.          0.          0.          0.          0.        ]
  [  0.9921875   0.00781252  0.          0.          0.        ]]

 [[ 14.         34.         35.         17.         38.        ]
  [ 34.         38.         25.         35.         15.        ]
  [  1.          4.          2.          0.          5.        ]
  [ 11.         21.         30.         27.          7.        ]
  [  7.         12.         40.         11.          8.        ]]]
```

The left column is the top pick. So from the first image in it correctly guessed 14 which represented stop sign. The 2nd row it guessed 34 which is turn left ahead, this was wrong, the correct answer was 28 which stood for children crossing. I pulled an image of turn left ahead from the internet, the color and shape are the same, which could be why the model chose 34.



The 3rd row it correctly guessed the speed limit 30 which was represented by 1, it is also no surprise that the other top 5 signs it guessed were the other speed limit signs from that dataset. They all have the same shape and color, the only thing different are the numbers. The fourth row it guessed right of way next intersection correctly as 11. I included the second guess double curve ahead which was 21. It makes sense that it considered this sign since the shape and color are similar.

The 5th and final row it incorrectly guessed 7 when the tested sign was roundabout which was denoted as 40. Sign 7 which was speed limit 100 has the same shape as the roundabout sign. But at least 40 appears as the models 3rd guess.



From the histogram we can see that 7 along with 40 has very few examples. This could have been the main problem with the misclassification of 7 and 40, there just were not that many examples to learn from. Also from the histogram the other sign it got wrong on the 2nd row a similar argument could be made on why it got the sign wrong.

Final Remarks

It is also worth mentioning when I deleted my checkpoint data and retrained my model and tested the same five images it once got 2 out of 5 signs correct. It seems that the initial randomization of weights did flip one of the classifications to some other convergence point related to that sign. Perhaps a local minima that it got stuck in? It is hard to recreate this result to explore it but maybe additional epochs helped it escape this hole which is why I cannot reliably recreate it.