German sign classification using deep learning

In this project we will go over classifying German traffic signs using deep learning with convolutional networks as a image classification. Experimented with different CNN's and stick with LeNet implementation because of the speed. Due to the trail and error in nature it took more than anticipated. There are more sophisticated CNN's to try with but didn't yield better results in my case.

Document organized as follows

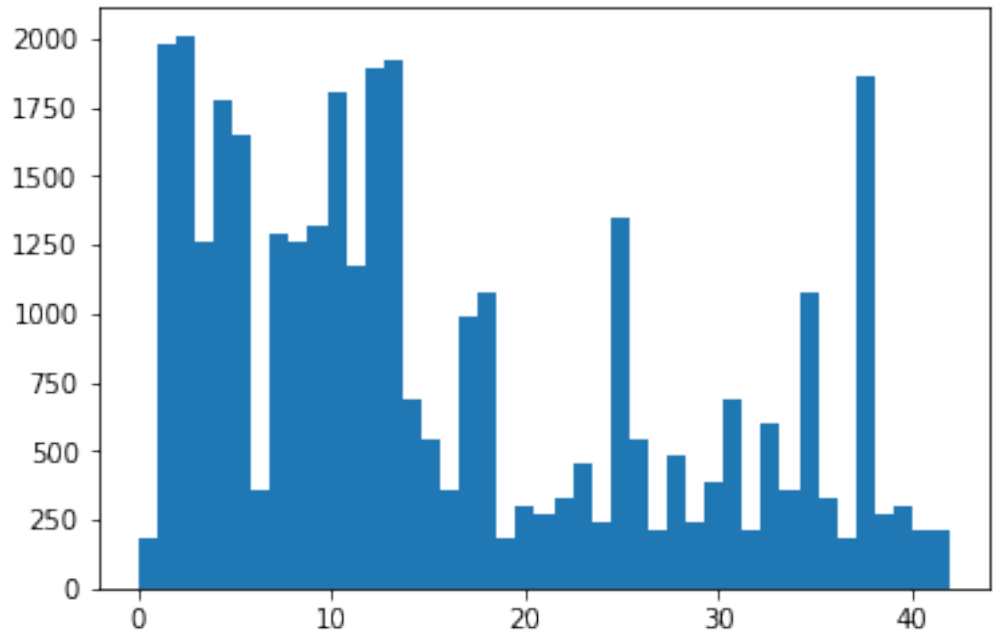
1) Data analysis
2) Preprocessing
3) Augmentation
4) Model architecture
5) Training
6) Performance analysis
7) Conclusion

Data Analysis

To familiarize the data printed sign total count and it's distribution as a bar chart. From the below table and bar graph, data has uneven distribution like 50km/h has highest total count and 20km/h has least total count.

	ClassId	SignName	Total
0	2	Speed limit (50km/h)	2010
1	1	Speed limit (30km/h)	1980
2	13	Yield	1920
3	12	Priority road	1890
4	38	Keep right	1860
5	10	No passing for vehicles over 3.5 metric tons	1800
6	4	Speed limit (70km/h)	1770
7	5	Speed limit (80km/h)	1650
8	25	Road work	1350
9	9	No passing	1320

10	7	Speed limit (100km/h)	1290
11	3	Speed limit (60km/h)	1260
12	8	Speed limit (120km/h)	1260
13	11	Right-of-way at the next intersection	1170
14	35	Ahead only	1080
15	18	General caution	1080
16	17	No entry	990
17	31	Wild animals crossing	690
18	14	Stop	690
19	33	Turn right ahead	599
20	15	No vehicles	540
21	26	Traffic signals	540
22	28	Children crossing	480
23	23	Slippery road	450
24	30	Beware of ice/snow	390
25	16	Vehicles over 3.5 metric tons prohibited	360
26	34	Turn left ahead	360
27	6	End of speed limit (80km/h)	360
28	36	Go straight or right	330
29	22	Bumpy road	330
30	40	Roundabout mandatory	300
31	20	Dangerous curve to the right	300
32	39	Keep left	270
33	21	Double curve	270
34	29	Bicycles crossing	240
35	24	Road narrows on the right	240
36	41	End of no passing	210
37	42	End of no passing by vehicles over 3.5 metric ...	210
38	32	End of all speed and passing limits	210
39	27	Pedestrians	210
40	37	Go straight or left	180
41	19	Dangerous curve to the left	180
42	0	Speed limit (20km/h)	180



After looking at the distribution of the sign data, checked couple of random samples of images which has

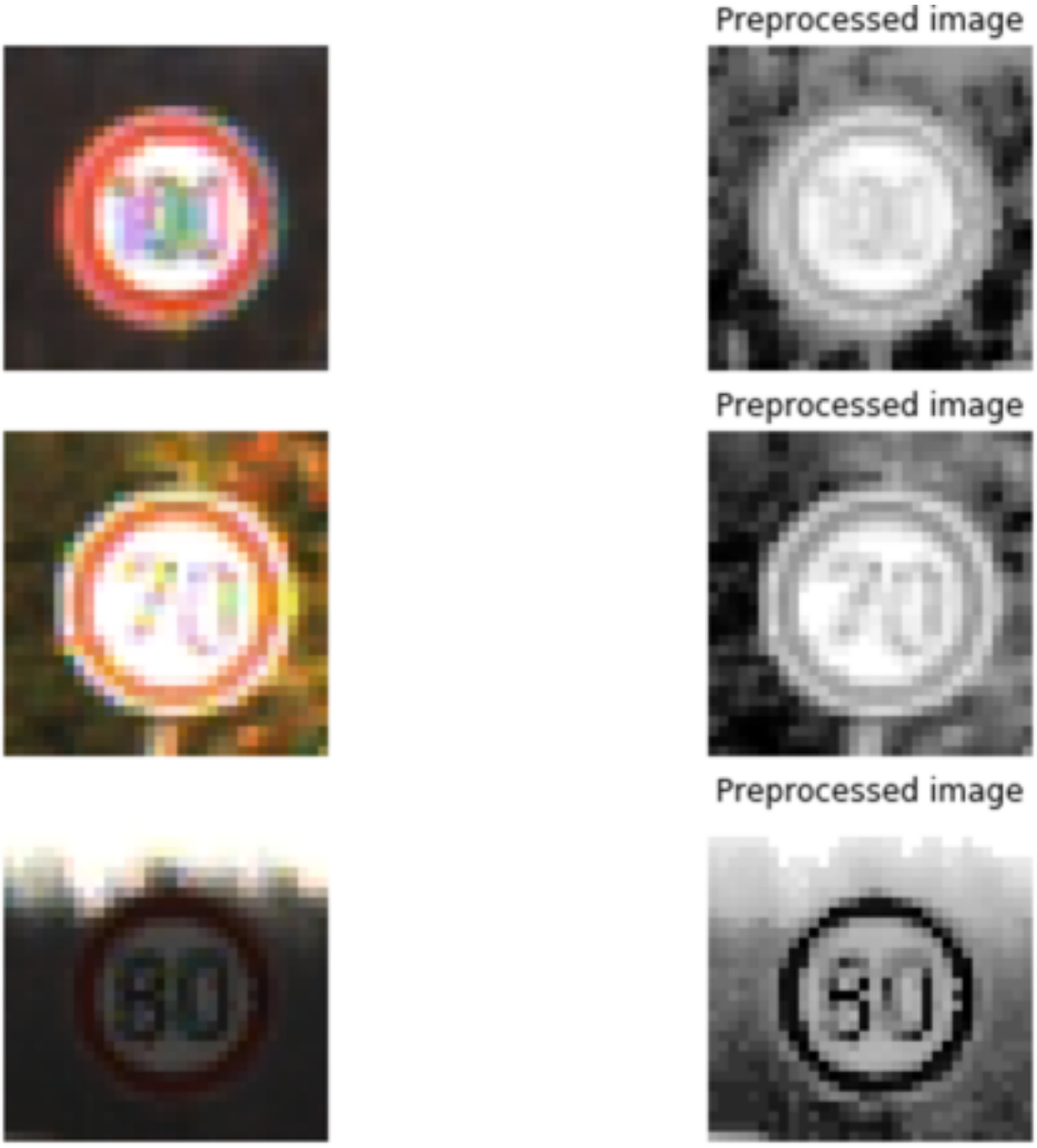
- different brightness.
- Equal in shape 32x32x3
- Different rotations

Preprocessing

In the preprocessing step used following technique

- converted original images to grayscale. It will speedup the process and also used LeNet model which works better with gray scale images
- equalized the brightness values in every image
- normalized the data based on mean. This helps weights should be smaller

Experimented with different strategies like translation, rotation and shearing. Didn't produce reasonable results. These are the sample images after preprocessing



Data Augmentation

Deep learning algorithms requires lot of data to perform well. Supplied traffic signs data from Udacity is not enough to achieve model accuracy. Augmented new data to the existing train set. Used following technique to produce more data

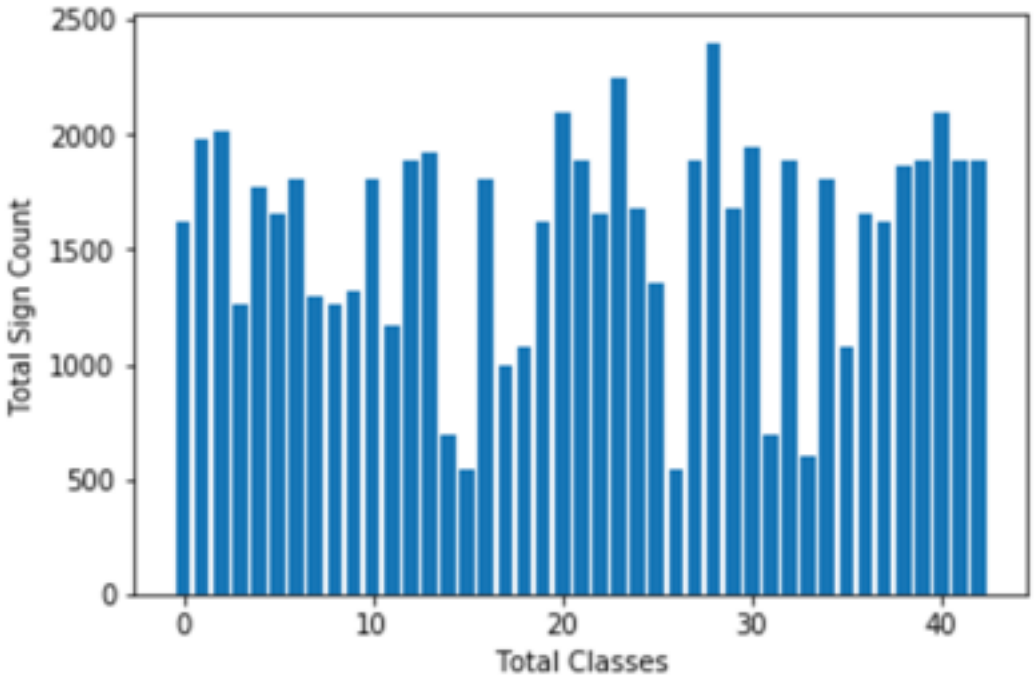
- Data is unbalanced like 50km/h has more data compare to 20km/h
- Calculated average, based on each sign classification count and augmented more data which has below average. Did not add any data to the signs which has more than average count
- Added more data using rotation or translation because other methods like shearing etc., didn't produce any improvement in model validation accuracy

Here are the summary of dataset including bar chart

```
Before pre process train data shape  (34799, 32, 32, 3)
Before pre process valid data shape  (4410, 32, 32, 3)
Before pre process test data shape  (12630, 32, 32, 3)

After pre process & normalize train data shape (34799, 32, 32,
1)
After pre process & normalize valid data shape  (4410, 32, 32,
1)
After pre process & normalize test data shape  (12630, 32, 32,
1)

Augmented data shape  (33000, 32, 32, 1)
```



```
New training size: 67799  labels size: 67799
```

Model architecture

Experimented with many models and finally settle down with LetNet. If I add more Conv layers computation time is quite high. Due to the nature of dataset LeNet can produce good results but not great results. Used two layers of convolution with max pooling and 3 fully connected layers

- Layer 1: Convolution with 5x5 filter, stride 1, depth 16, max pool stride 2
- Layer 2: Convolution with 5x5 filter, stride 1, depth 32, max pool stride 2
- Flatten layer and used dropout with probability 0.8
- Layer 3: FC layer with 128 nodes
- Layer 4: FC layer with 64 nodes
- Layer 5: Output with 43 nodes

Before I conclude above architecture I conducted following experiments

- Used Udacity implementation of LeNet and it produced validation accuracy <0.72. Then I realized it could be because of the dataset. Increased dataset size from 67K to 170K which produced very bad results
- I experimented with 3 convolution layers with 3x3 filter with 32 depth and added 3 more convolution layers with 3x3 filter with 64 depth. Which didn't yield any good results. Then I realized adding more layers taking time to test and iterations.
- Then I setup a goal such a way that, stick to LeNet and start experimenting different parameters. Above architecture yielded good results after adding dropout with reasonable amount of testing time.

After reading papers on CNN's added dropout which started yielding good results. Most of the parameters tuned based on trail and error.

Training

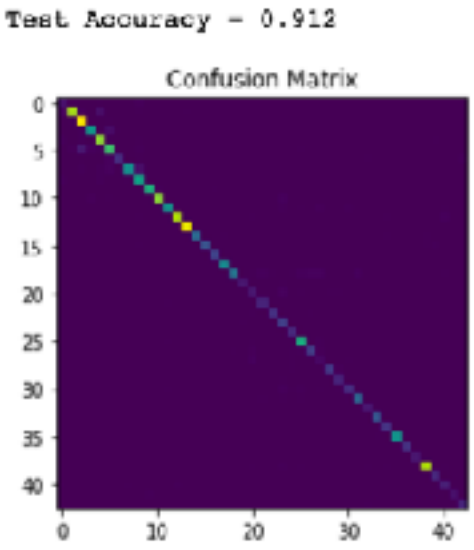
Trained with following parameters

- Used Adam Optimizer, didn't get time to experiment with other optimizers
- Tried different epochs and settle down with 20
- Experimented different learning rates finally picked 0.0009
- Added L-2 regularization to avoid overfitting but after commenting out regularization it didn't change the validation accuracy

Compare to other optimizers Adam Optimizer converges faster but didn't experiment other optimizers. In my case validation accuracy did not improve beyond 20 with current LeNet architecture implementation.

Performance analysis

At some point I was able to achieve 0.96 validation accuracy with different trail and error methods, don't know which parameter configuration achieved it. Finally I was able to achieve 0.94 validation accuracy and 0.912 test accuracy.



By looking at confusion matrix majority of errors located in top left which are related to speed limits. That means it can able to detect speed limits but trouble identifying numbers on the sign. Also dataset is biased towards couple of signs rather than equal distribution.

Downloaded new traffic sign images and preprocessed it

End results on predicting new images are not that great. It was able to predict only stop sign closely other things are way off due to background color after preprocessing these images. Also model was able to predict closely if the sign exists in train dataset.

Conclusion

In this project end up reading lot of papers especially deep learning related to CNN's. Only thing I am puzzled on trial and error approach of data analysis and tuning CNN params. I could have allocated more time on understanding these parameters to tune the models. I really thank you all the Udacity staff members to put together nice videos and projects on deep learning.