Build a Traffic Sign Recognition Project

Louis Pienaar

02 March 2017

The goals / steps of this project are the following:

- 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

Data Set Summary & Exploration

1. Data set summary

The code for this step is contained in the fourth code cell of the IPython notebook.

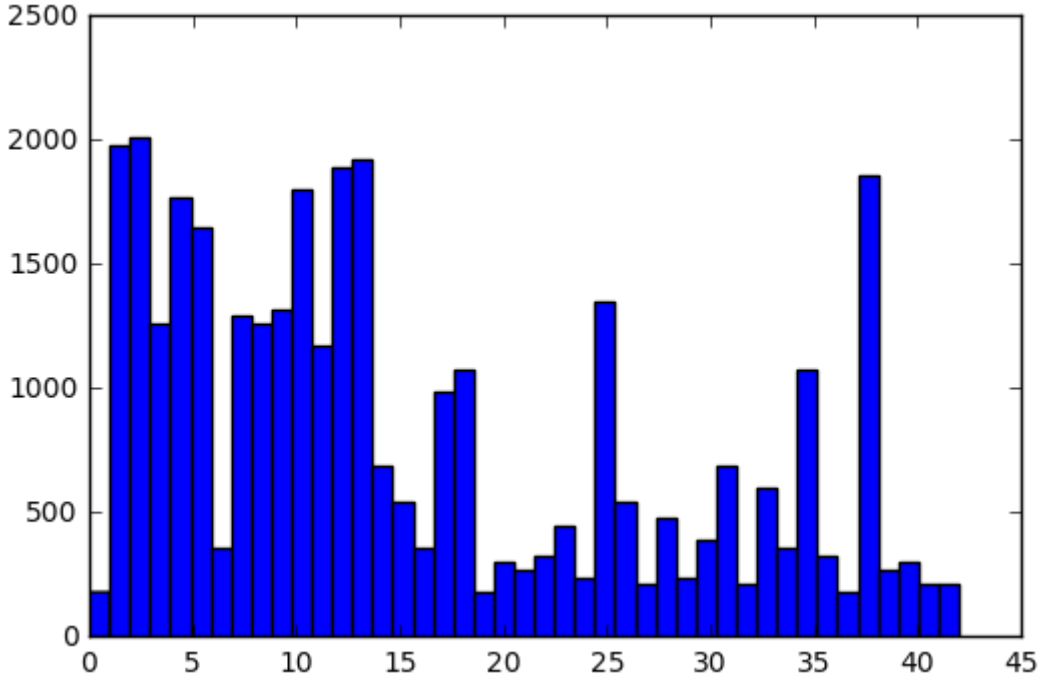
- The size of training set is 34799
- The size of test set is 12630
- The shape of a traffic sign image is (32,32,3)
- The number of unique classes/labels in the data set is 43

2. Exploratory visualisations.

The code for this step is contained in the fifth code cell of the IPython notebook.

Here is an exploratory visualization of the data set. We first have a look at a random set of 20 images from the data. This glimpse shows as that the images are often unclear, dark, and often full of artifacts. A good challenge for a neural network.

This bar chart shows the number of images for each of the 43 classes. It is clear that some classes have very few images. A neural network requires large amounts of data to perform well, hence the fact that some classes have so few images, needs to be catered for.



Design and Test a Model Architecture

1. Image pre-processing

The success of the model depends largely on the pre-processing done. There are some really good literature available on the possible networks you can use to achieve results, but the art of modelling lies mostly in you data preparation.

The first decision I made was to not convert to gray scale. My reasoning for this was that a good network would be able to use the data of three input channels better than data with one input channel only. I did not test this assumption and would only re-visit this decision if the model is not performing.

In code block 7, I consolidated functions to do image augmentations for the purpose of creating more training data. These augmentations included rotating the image and applying affine and translation transformations using random parameters within a given range.

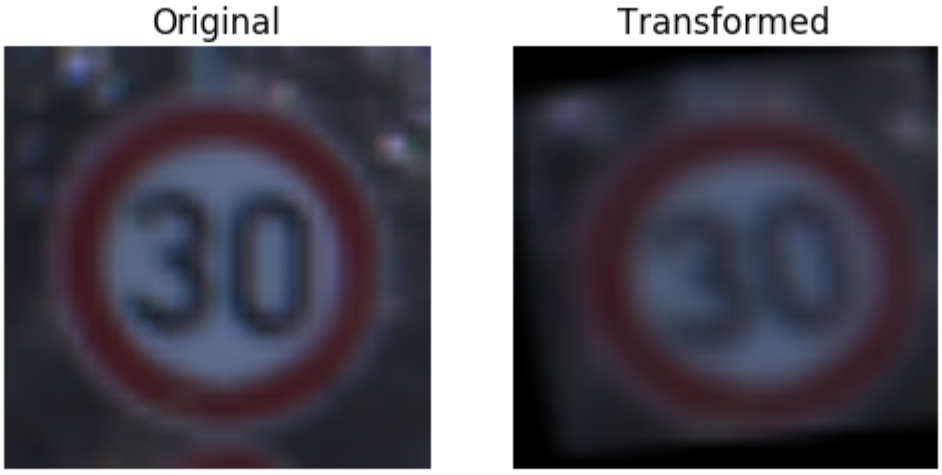


Image augmentation is fairly CPU intensive. A future improvement would be to research a method to do this quicker, potentially even using GPU to handle augmentations. Because of the CPU limitation, I resorted to adding a set 400 augmented images per class to the training data. The code for this are in code block 9.

Next I tackled the problem of intensity, luminosity or brightness. I was shocked to see how many images are extremely dark and almost impossible to distinguish. I assume the network might have similar problems. Because of this I experimented with Contrast Limited Adaptive histogram equalization CLAHE. Very simply put, CLAHE is good at adjusting limits for parts of the image instead of the entire image. However, I included simple RGB histogram equalization for comparison.

Here is an example of the process, the cl picture is a representation of the adjusted v channel of the HSV formatted image:

After running through a few examples, I decided to not use the CLAHE method as it wasn't performing as expected. I therefore only did histogram equalization on each of the channels in the RGB image. This was done in code block 14 and 15.

Here is a set of 20 images after the RGB histogram equalization:

Lastly I applied min max scaling by simply dividing by 255 multiplying by 0.8 and adding 0.1. This scales the image data between 0.1 and 0.9. The code for this is in code block 20.

Using a simple mean comparison, you can see that the data is fairly similar between the train, test and validation sets. However, the extra images added did seem to lower the overall mean for the validation set. Something to consider for future model improvements.

Mean(Train) = 0.492455761192 Mean(Test) = 0.514030070114 Mean(Valid) = 0.513327151926

I chose min max scaling purely from tips taken during the initial lessons.

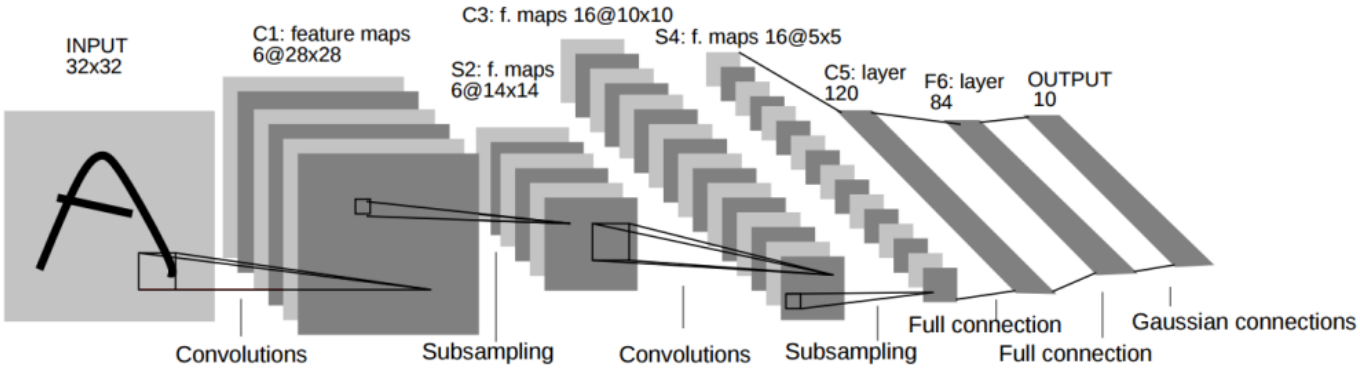
2. Additional data generated, and the Train / Test / Validation sets

As mentioned before, I created 400 additional images for each of the 43 classes, totaling to an extra 17200 images in the training set. The data provided for this version of the project already split the images into train, test and validation.

3. Model architecture.

I spend too many hours experimenting with different networks and their performance. My favorite being the network proposed by Vivek Yadav achieving 98.8% accuracy. https://github.com/vxy10 (https://github.com/vxy10) However, I learned allot. After all that I reverted to the original Lenet55 pipeline as I found it to be a better learning experience to code from that framework.

Here is a picture of the original Lenet5 network.



Thanks to the excellent training provided by Udacity I was able to improve this network by doing the following:

- Change input layer to three channels
- Add dropout layers
- Increase the fully connected layer widths

The increase in the fully connected layer widths was done as it improved the accuracy from about 0.92 to 0.95 on the validation set (The validation set is used during training and the test set was used as the final test)

My model also include l2-regularization. Both the regularization and the dropout layers reduces the risk of over-training.

The code for my final model is located in the 26ht code cell of the ipython notebook.

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution 5x5x3	1x1 stride, valid padding, outputs 28x28x6
RELU	

Layer	Description
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5x16	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flatten	Outputs 400
Fully connected	Outputs 200
RELU	
Dropout	
Fully connected	Outputs 100
RELU	
Dropout	
Fully connected	Outputs 43
Softmax cross entropy	

4. Model Training

The code for training the model is located in the 27th and 28th code cell of the ipython notebook.

To train the model, I used an Adam optimizer with a learning rate of 1e-3. I added regularizers to the loss operation to help prevent over fitting. Most of my parameter choices were driven by the need to prevent over fitting. The next section will hopefully show you the fruits of these choices.

The final model was trained on a batch size of 75, mu of 0 and sigma of 0.05 and epoch of 35. I did not code an early termination into the learning procedure, but I did keep record of the accuracy measures during training and performed a manual termination. Form this I learned that the model starts over fitting form about 35 epochs onward.

5. Model testing approach

As mentioned earlier, I tried numerous architectures. Some of which I struggled to get to work at all. None of my attempts got close to a good accuracy or close to the expected accuracy. However, I did learn allot form these other architectures. Here is a list of improvements that I want to add to my current architecture when I have a chance:

- Add 3 1x1x3 convolution layers at the start of the graph to act as initial filters
- Expand the size of the other convolution layers
- Link the convolution layers in parallel and series to the initial flatten layer. Giving the network the opportunity to choose if it wants to use the other conv layers.
- Do more hyper parameter tuning

The Lenet5 architecture proves to be really good for this task. After the few adjustments made it achieved an accuracy level that I was super happy with. It also managed to perform really well on the holdout test set which showed to me that there was no over fitting.

My biggest concern on my model is highlighted in the next section

The code for calculating the accuracy of the model is located in the of the Ipython notebook.

My final model results were: * training set accuracy of 0.994 (based on the original training set before adding augmented images) * validation set accuracy of 0.956 * test set accuracy of 0.932

Test a Model on New Images

1. 5 German traffic signs from the internet

Here are five German traffic signs that I found on the web:

I tried to get a mixture of images in terms of quality. I had this theory that perfect signs taken from the web, IE signs with no background and perfect colors might not be easily classified. I was also worried about warping due to the fact that I adjusted the picture to fit in 32x32.

I expected that if one of the images classifications would fail, it would be the pedestrian, as this is a near perfect image of the sign and the model probably has not seen that sign this clearly yet.

However, the model classified all but the yield sign correctly. Even the 30 speed limit sign.

4 out of 5 = 80% accuracy.

I have no idea why the yield sign wasn't classified correctly. My first guess would be that the sign covers too much of the 32x32 space. In other words if the image is zoomed out and filled with black space, it would have been classified correctly. This idea lead me along a path to a few ideas on how to improve the model.

2. Model predictions on my own images

The code for making predictions on my final model is located in the 30th code cell of the Ipython notebook.

Here are the results of the prediction:

30speed
Speed limit (30km/h)
Speed limit (20km/h)
Speed limit (120km/h)
Speed limit (70km/h)
Speed limit (50km/h)

Pedestrians
Pedestrians
General caution
Right-of-way at the next intersection
Traffic signals
Bumpy road

priorityroad
Priority road
No entry
Stop
No passing for vehicles over 3.5 metric tons
Go straight or left

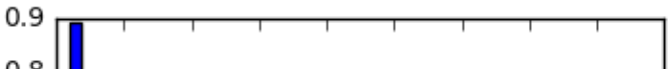
roadwork
Road work
Road narrows on the right
General caution
Right-of-way at the next intersection
Beware of ice/snow

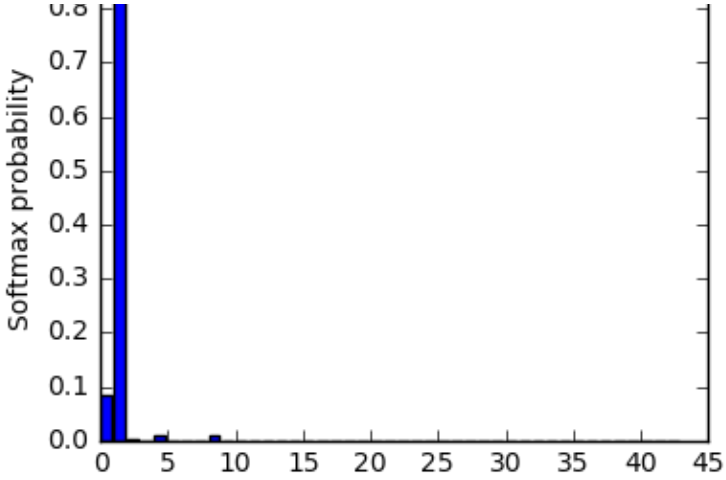
Sign4
Speed limit (120km/h)
Speed limit (70km/h)
No vehicles
Yield
Speed limit (30km/h)

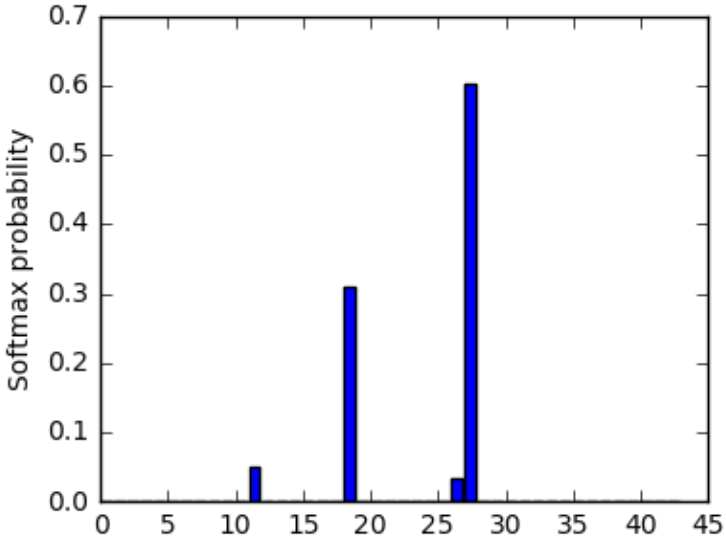
The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This is acceptable compared against the Test set accuracy of 90%+ . However I was expecting a 100% accuracy ####3. Softmax analysis

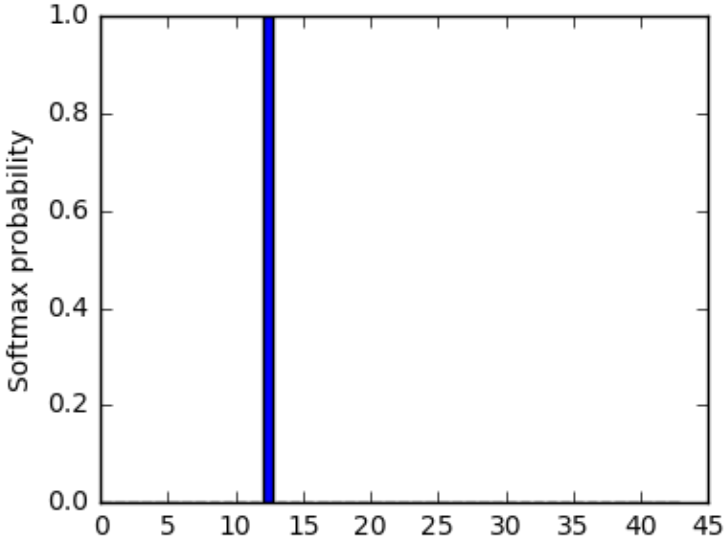
Softmax probabilities gives us an idea on how certain the model is about some of it classifications. Form experience I know that it is common for neural networks to often have close to 100% certainty on it’s predictions.

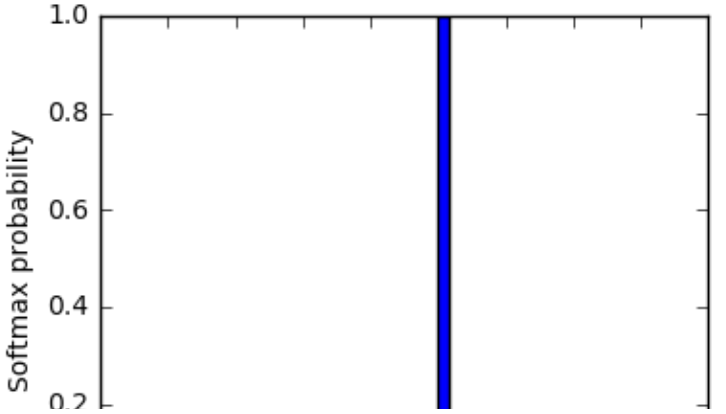
Here is the softmax probabilities for the 5 new images:

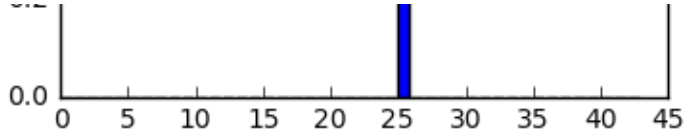


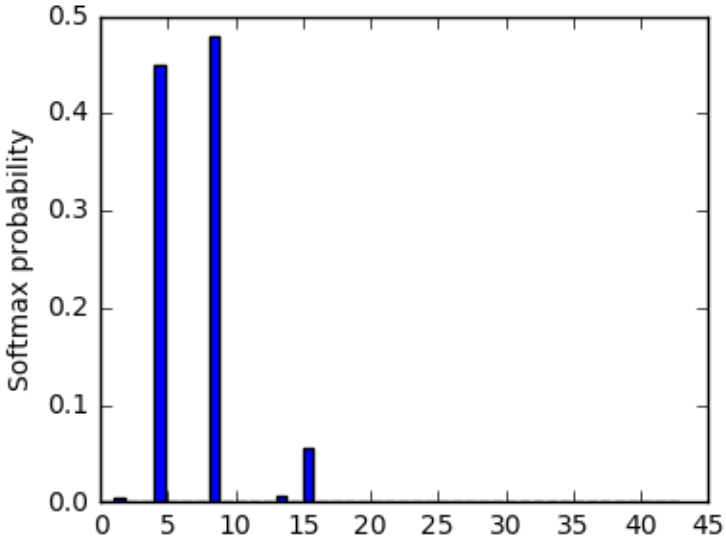

Softmax probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0 5 10 15 20 25 30 35 40 45


Softmax probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0 5 10 15 20 25 30 35 40 45


Softmax probability
0.0
0.2
0.4
0.6
0.8
1.0
0 5 10 15 20 25 30 35 40 45


Softmax probability
0.2
0.4
0.6
0.8
1.0





For three of the 5 (Roadwork, Priority road and 30 speed limit), the model had near 100% certainty on the classifications. Amazing.

On the Pedestrian, you can notice that it was not that certain. On the Yield it is also has some uncertainty albeit that the yield softmax probability was extremely low

4. Conclusions and suggestions

I can not wait to show colleagues and friends how powerful neural networks done right are. The level of accuracy is astounding and having seen other solutions before even better than mine give me allot of comfort in data science.

The Lenet5 architecture is old and newer architectures are slowly being build that improves on the Lenet architecture. That being said, here are some ideas that I would like to test to improve accuracy:

- Bagging - A form of bagging that combines softmax probabilities from vastly different neural networks
- Multiple predictions on the actual image, randomly augmented. For example. My yield sign as is were incorrectly classified, however, if i made 5 different augmented images form this image, then the model would have been likely to classify the 5 correctly on average

I have not seen many examples of my two ideas posted here, I would love to test them and publish them.

5. References

During this project I have researched methods and code from the folowing resources:

- https://chatbotslife.com/german-sign-classification-using-deep-learning-neural-networks-98-8-solution-d05656bf51ad#.5r9pm6flu (https://chatbotslife.com/german-sign-classification-using-deep-learning-neural-networks-98-8-solution-d05656bf51ad#.5r9pm6flu)
- https://github.com/jeremy-shannon/CarND-Traffic-Sign-Classifier-Project/blob/master/Traffic_Sign_Classifier.ipynb (https://github.com/jeremy-shannon/CarND-Traffic-Sign-Classifier-Project/blob/master/Traffic_Sign_Classifier.ipynb)
- http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html (http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html)
- https://sumitbinnani.github.io/CarND-Traffic-Sign-Classifier-Project/ (https://sumitbinnani.github.io/CarND-Traffic-Sign-Classifier-Project/)
- https://github.com/mvirgo/Traffic-Sign-Classifier/blob/master/Traffic_Sign_Classifier.ipynb (https://github.com/mvirgo/Traffic-Sign-Classifier/blob/master/Traffic_Sign_Classifier.ipynb)