Build a Traffic Sign Recognition Project

Sraven Saradhi

The goals / steps of this project are the following:

- Load the data set (see below for links to the project 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

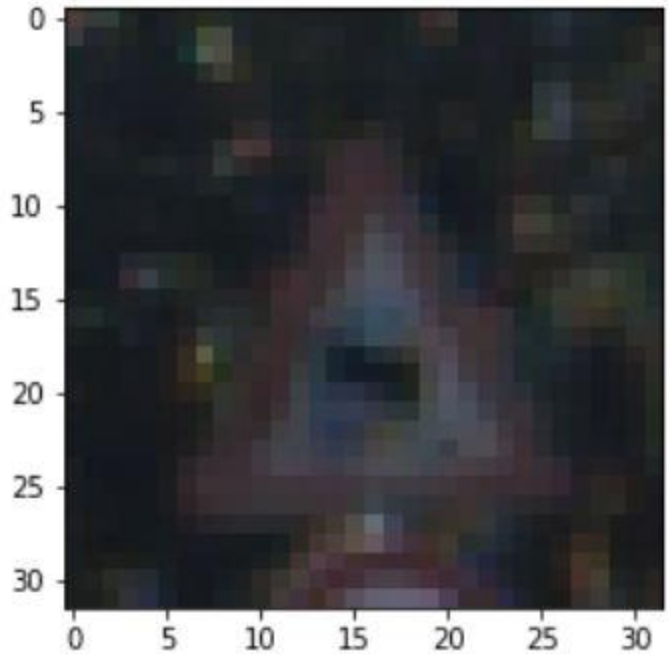
The key points of the data were extracted from using a mix of numpy functions and general python index functions. The key functions used were the length function, the set function and the shape function. The set function removes duplicates, the length function gives the number of examples.

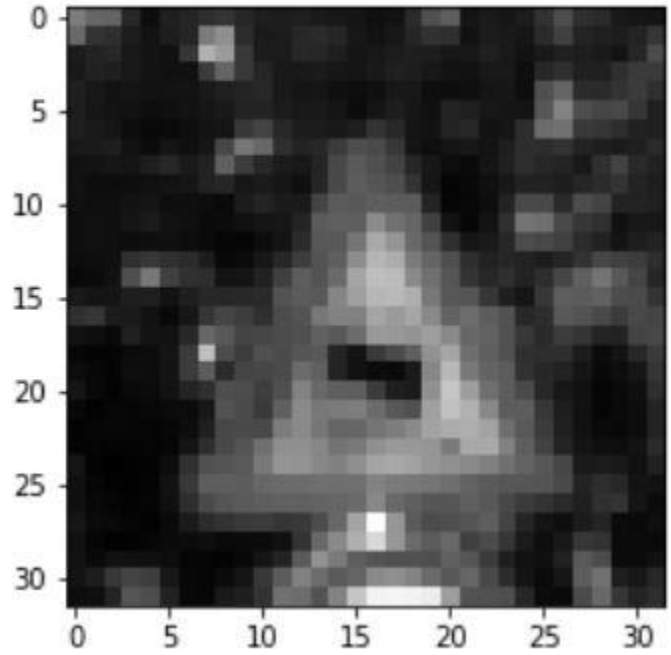
```
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

For exploration, I wanted to visualize how the traffic signs looked along with their respective labels. To further showcase data, I plan to create a histogram to show a count of each sign and other factors. For this project, I wanted to keep it simple with a visualization of the signal along with the x and y axis.

Design and Test a Model Architecture

For the design of the preprocessing I decided to use gray-scaling to make it easier for the algorithm to classify data by removing parameters. Initially I wanted to keep the rgb data but found the accuracy lacking, to simplify the process and improve accuracy, I decided to use grayscale. I then normalize the data to put the data between 0 to 1. This is mainly to standardize the data and ensure that all data can be comparable. Both of these techniques along with changes to the model architecture which I will discuss in the following paragraph improved data accuracy.





The grayscale shows more definition in terms of sign border.

When dealing with the CNN, I decided to use the Lnet architecture with changes to the number of classes and an addition of another function known as the tf.nn.dropout after the tf.nn.relu function. The dropout function randomly sets elements to 0 to prevent overfitting of data.

Finally, for model training the parameters of EPOCH were changed to inclined more baskets of data, the batch size was made smaller and the learning rate was tweaked to be close to 0.001. Making the small changes to the learning rate made small changes to the accuracy to get it into an acceptable amount.

The overall architecture is below:

Layer	Description
Input	32x32x1, grayscale
Convolution	Output = 28x28x6, stride = 1x1
Relu	
Max pooling	ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID'
Convolutional.	Output = 10x10x16. stride = 1x1
Relu	
Max pooling	ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID'
flatten	Output = 400.
Relu	
Dropout	Keep_prob = 0.5
Fully Connected	Output = 120.
Relu	
Dropout	Keep_prob = 0.5
Fully Connected	Output = 84
Relu	
Dropout	Keep_prob = 0.5
Fully Connected	Output = 43.

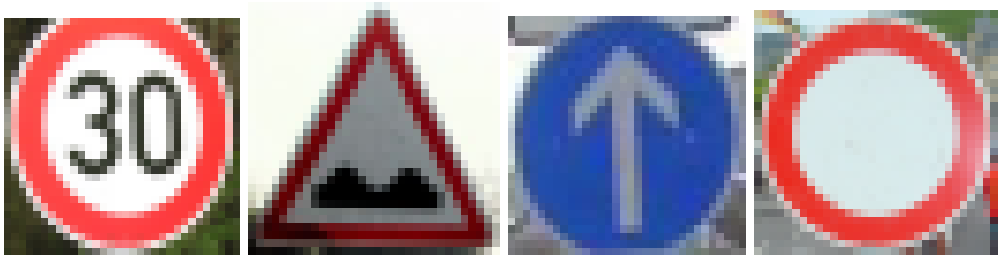
My final model results were:
Training set accuracy: over 96%
validation set accuracy: 94.2%
test set accuracy: 91.8%

The process to get to this final model was an iterative process:

- The first model was the Lnet model used in the lab with no preprocessing. This yielded an error of around 80%. The Lnet model was chosen due to it's simple yet effective nature. It gave a lot of flexibility in tuning and adding more activation functions.
- The second model was the Lnet model which normalizes the data in the preprocessing phase. I combined this with changes to the changes in learning rate and number of EPOCHS. This gave a max validation percent of around 89%. The reasoning was to standardize the data and tune the parameters, increasing the EPOCHS, increasing the batches and showing the steady changes in accuracy. The learning rate was tuned accordingly.
- The 3rd model included a grayscale which improved the accuracy further but still didn't hit the 93% mark. It got to about 91%. Grayscale improved accuracy because it simplified the parameters and also defined the object boundary better as can be seen from the image comparison above. I added another layer to spread the data more.
- The final model included a dropout function along with the changes to the batch size. EPOCH and learning rate adjustments were also included. This accounted for overfitting, simplified classification and gave a percent of above 93%.

Test the model on New images

I chose six images of german traffic signs by googling some images. The images below are the six I picked.



The first image might be difficult to classify because of the number on the center which can be of different curves and can be confused for lines.

The second image is of a triangular shape with curvy features which could pose problems.

The third image is of a different color and darkers which could cause problems in classifying.

The fourth image is simple but may cause problems when gray-scaled

The fifth image has 2 arrows and is made of complex curves, this could potentially throw off the algorithm.

The last image may be confused with a downward arrow which could cause errors.

The table showing the actual sign along with the predicted sign is given below:

Actual	Predicted
Speed limit 30	Go straight or left
Road with bumps	Road with bumps
Ahead	Ahead
No vehicles allowed	No vehicles allowed
Go straight or turn left	Go straight or turn left
Caution ahead	Caution ahead

The test accuracy came out to be around 83%, it was able to accurately guess 5 out of the 6 test images. The accuracy of the new predictions is comparable to that on the original test set. With more data the percentage would surely increase to about that of the actual test set.

The code for making predictions on my final model is located in the 14th cell of the Ipython notebook.

For the first image, the model is relatively sure that this is a Go straight or left sign (probability of 0.93), and the image contains a stop sign. The top five soft max probabilities were

Probability	Prediction
9.29013252e-01	Go straight or turn left
6.75137341e-02	Speed limit (30km/h)
2.39792792e-03	Keep left
9.80600133e-04	Speed limit (50km/h)
6.29604328e-05	Roundabout mandatory

For the second image, the model is relatively sure that this is a Bumpy road sign (probability of 0.99), and the image contains a Bumpy road sign. The top five soft max probabilities were

Probability	Prediction
9.97808039e-01	Bumpy road
1.51423411e-03	Speed limit (30km/h)
6.26981433e-04	Keep left
4.66778256e-05	Speed limit (50km/h)
3.93213440e-06	Roundabout mandatory

For the third image, the model is relatively sure that this is a Ahead only sign (probability of 0.99), and the image contains a Ahead only sign. The top five soft max probabilities were

Probability	Prediction
9.99999523e-01	Ahead only
4.23956891e-07	Go straight or right

6.35810693e-08	Turn left ahead
1.31565236e-09	Dangerous curve to the right
3.51861124e-10	No passing

For the fourth image, the model is relatively sure that this is a Yield sign (probability of 0.66), and the image contains a Yield sign. The top five soft max probabilities were

Probability	Prediction
6.68615937e-01	Yield
1.09330662e-01	No vehicles
9.25795212e-02	Turn left ahead
7.25905448e-02	Ahead only
3.49594839e-02	Priority road

For the fifth image, the model is relatively sure that this is a Go straight or left sign (probability of 0.70), and the image contains a Go straight or left sign. The top five soft max probabilities were

Probability	Prediction
7.02307880e-01	Go straight or left
1.04292497e-01	Speed limit (30km/h)
9.36722234e-02	Roundabout mandatory
2.60708071e-02	Pedestrians
2.34880149e-02	Double curve

For the sixth image, the model is relatively sure that this is a General caution sign (probability of 0.99), and the image contains a General caution sign. The top five soft max probabilities were

Probability	Prediction
9.96303916e-01	General caution

3.69613874e-03	Traffic signals
1.98581845e-10	Pedestrians
1.63471488e-21	Road narrows on the right
9.60644419e-28	Bumpy road