Traffic Sign Recognition

Build a Traffic Sign Recognition Project

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

Rubric Points

Here I will consider the rubric points (https://review.udacity.com/#!/rubrics/481/view) individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! also explanations are in project code (https://github.com/adarshakb/CarND-Traffic-Sign-Classifier-Project)

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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

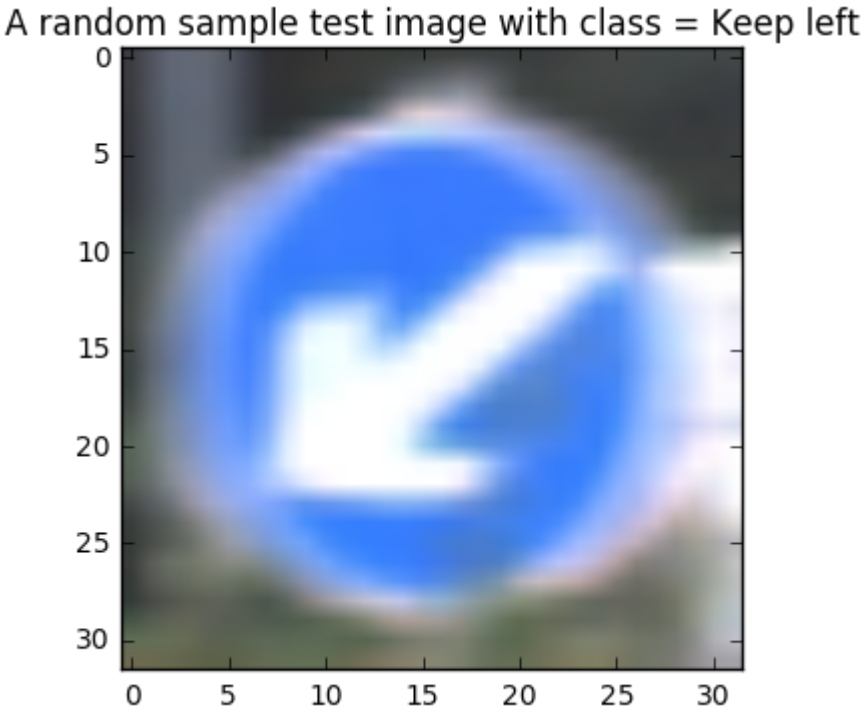
I used the pandas library to calculate summary statistics of the traffic signs data set:

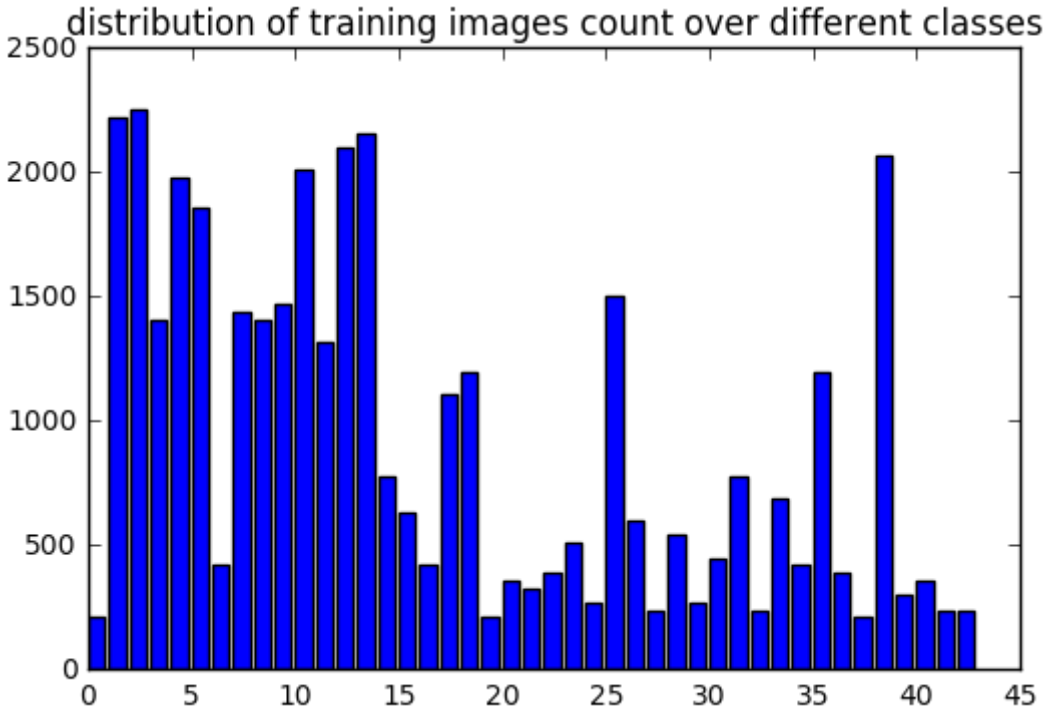
- Number of training examples = 39209
- Number of testing examples = 12630
- Image data shape = (32, 32, 3)
- Number of classes = 43

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

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

Here is an exploratory visualization of the data set. It is a bar chart showing how the data ...





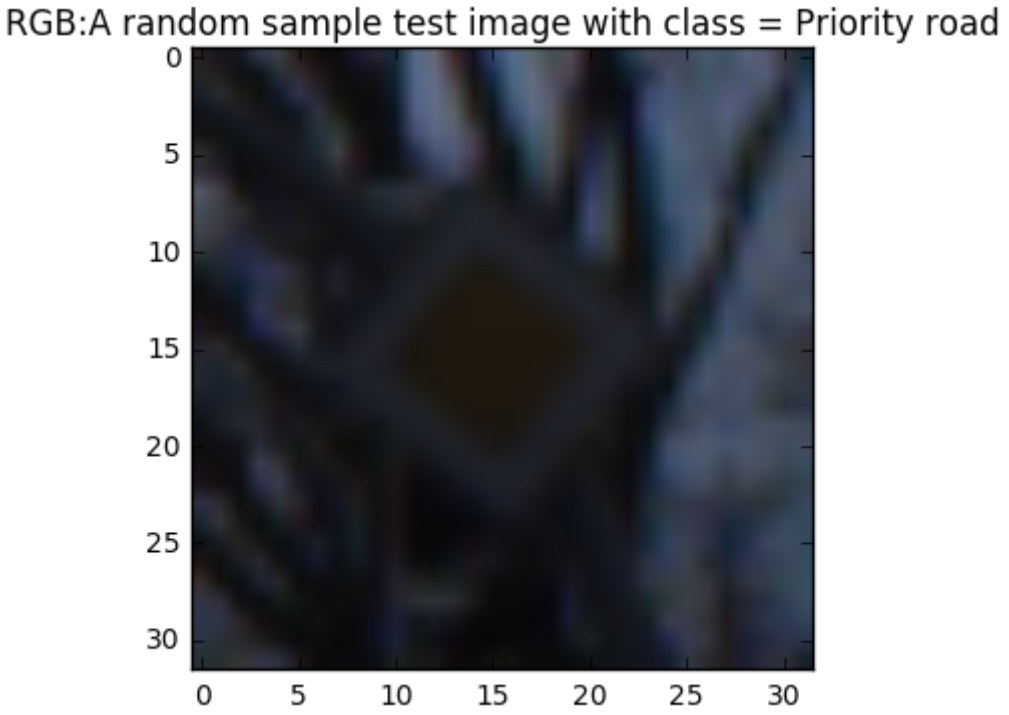
Design and Test a Model Architecture

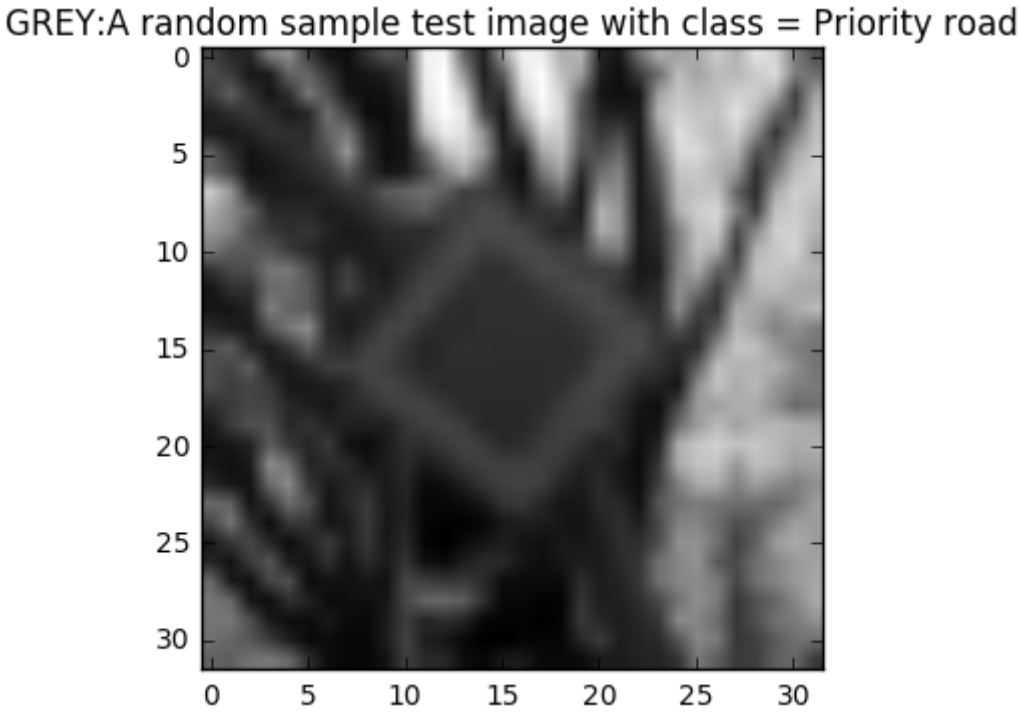
1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

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

As a first step, I decided to convert the images to grayscale because it will reduce the number of channels and hence the number of variables finally in the network. This means we can train better with lesser data. However we will loose any information which might require a "color" component. We donot anticipate this problem here

Here is an example of a traffic sign image before and after grayscaling.





As a last step, I normalized the image data because this will constraint the inputs between 0 and 1 which is better for the classifier so that there is no wide variations in the data

2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

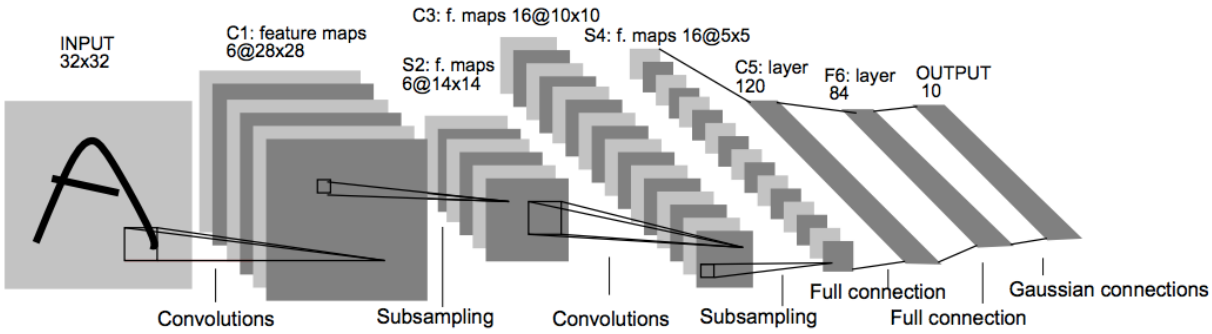
The code for splitting the data into training and validation sets is contained in the fifth code cell of the IPython notebook.

To cross validate my model, I randomly split the training data into a training set and validation set. I did this by using the buily in train_test_split of sklearn which shuffles the data also.

My final training set had 31367 number of images. My validation set and test set had 7842 and 12630 number of images.

3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

My code is based on the LeNet architecture (**with additional modifications noted below**)



(Image for clarity. Not the final model)

The code for my final model is located in the 6th cell of the ipython notebook.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Gray scale image
Convolution	1x1 stride, same padding output 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution	Input 14x14x6 and output 10x10x16.
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flatten	convert to 1D array - 400
Reduce .	400 to 120
RELU	

Layer	Description
DROPOUT	0.5 probablity incase of training
Reduce .	120 to 84
RELU	
DROPOUT	0.5 probablity incase of training
Reduce .	84 to 43 **final**
Softmax	apply softmax on logits

Training with Adam Optimizer (based on gradient decent)

4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

The code for training the model is located after the model architecture cell of the ipython notebook.

To train the model, I used a batch size of 128 to highly parallize the process. I also kept 100 EPOCHS as the upper limit. However I also limit it based on validation set accuracy. I only run more EPOCHS if we are still *learning at a rate >= 0*. This means we donot overfit the data w.r.t training data. The learning rate has been kept at 0.0003 so that the network is unsure at first and then becomes more confident. I have kept the default LeNet mu & signma hyperparameters because I found them optimal. I also choose the mean crossentropy and gave it to minimize for the optimizer

5. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

The code for calculating the accuracy of the model is located in the 16,17th cell of the Ipython notebook.

My final model results were:

- validation set accuracy of 95.575%
- test set accuracy of 89.7%

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?
  - I tried the architecture without Droupout & Also without restricting the EPOCHS if we are still learning*.
- What were some problems with the initial architecture?
  - without dropout the results were wrong when there was a little noise in the images
  - wihtout EPOCH restriction, even tho the validation data accuracy was good it would overfit and get less in test data
- How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling,

dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to over fitting or under fitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.
    - explained above
- Which parameters were tuned? How were they adjusted and why?
    - I tuned the learning rate. I lowered it down to get better accuracy prediction so that the network can start out unceratin and not overfit
- What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?
    - Dropouts help us to have redundant network connections.
    - ReLU help us learn complex functions in the network rather than a simple linear function
    - Maxpooling help us identify which pixel in a region is dominant .. such as lines or boundaries

If a well known architecture was chosen:

- What architecture was chosen?
    - LeNet modification
- Why did you believe it would be relevant to the traffic sign application?
    - LeNet has been shown to work well to classify images
- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?
    - We have high accuracy on test set close to 90%. These are with certain limitations like having low number of examples in many of the classes for the training as seen the previous barchart analysis

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

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

- might be difficult to classify due to because it should be able to differentiate between 20/30/40/50/60 etc signs which all look almost similar (circle & 0 digit - about 80% of image are same0)

- might be difficult because there is some grains in the image
- its also similar to go straight

- the triangle is present in many other signs

- difficult to classify as triangle is present in many other types of signs

- might be difficult to classify due to because it should be able to differentiate between 20/30/40/50/60 etc signs which all look almost similar (circle & 0 digit - about 80% of image are same0)

UPDATE:

- If you look at the distribution shown in the last of this notebook, you will find that the model got it almost right but could not decipher the numbers. Looking at the distribution of the input training classes we can clearly see that the number of examples supplied for these are low. Thus the model has overfit for those which it found more examples.
- the accuracy on the captured images is 60% while it was 89% on the testing set thus It seems the model is overfitting.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

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

Here are the results of the prediction:

Image	Prediction
Speed limit (50km/h)	Speed limit (30km/h)
Go straight or left	Go straight or left
Road work	Road work
General caution	General caution
Speed limit (20km/h)	Speed limit (30km/h)

The model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%.

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the final 2 sections of the Ipython notebook.

For the first image, the model is relatively sure that this is a stop sign (probability of 0.6), and the image does contain a stop sign. The top five soft max probabilities were

```
[  7.68104970e-01,   2.05424443e-01,   7.58509990e-03,
          7.06370175e-03,   6.50352100e-03],
       [  9.99999642e-01,   3.79177806e-07,   5.58574591e-08,
          1.79112689e-08,   3.39173933e-09],
       [  9.86939788e-01,   1.04183871e-02,   8.82952765e-04,
          5.93022211e-04,   4.24549507e-04],
       [  9.98059690e-01,   1.01856072e-03,   7.53592234e-04,
          1.67402904e-04,   2.29664465e-07],
       [  8.63265514e-01,   1.35711610e-01,   9.94851231e-04,
          1.67907965e-05,   5.44629938e-06]], dtype=float32), indice
s=array([[ 1,  2,  5,  6, 12],
       [37, 39, 40,  4, 18],
       [25, 29, 23, 31, 30],
       [18, 26, 27, 24, 29],
       [ 1,  0, 12,  2, 13]],
```

Image 1: - it got confused between 5 & 3 :(

Probability	Prediction
.78	Speed limit (30km/h)
.20	Speed limit (50km/h)
.007	Speed limit (80km/h)
.007	End of speed limit (80km/h)
.0065	Priority road

Image 2:

Probability	Prediction
.99	Go straight or left
~.00	Keep left
~.00	Roundabout mandatory
~.00	Speed limit (70km/h)
~.00	General caution

Image 3:

Probability	Prediction
.98	Road work
.01	Bicycles crossing

Probability	Prediction
~.00	Slippery road
~.00	Wild animals crossing
~.00	Beware of ice/snow

Image 4:

Probability	Prediction
.99	General caution
.001	Traffic signals
~.00	Pedestrians
~.00	Road narrows on the right
~.00	Bicycles crossing

Image 5: It got confused between 2 & 3 :(

Probability	Prediction
.86	Speed limit (30km/h)
.13	Speed limit (20km/h)
.001	Priority road
~.00	Speed limit (50km/h)
~.00	Yield

In [ ]: