#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 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! and here is a link to my project code

###Data Set Summary & Exploration

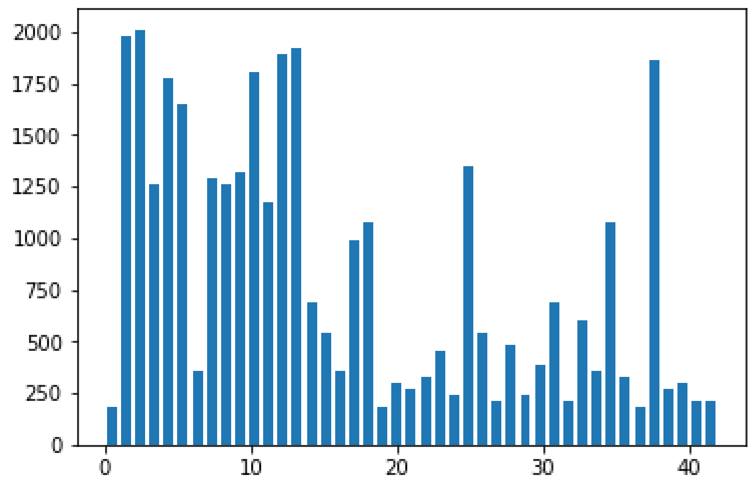
####1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

- The size of training set is - 34,799
- The size of the Udacity supplied validation set is - 4110
- The size of test set is - 12,630

- The shape of a traffic sign image is - 32x32x3
- The number of unique classes/labels in the data set is - 43

####2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing the frequency of given training data by label/class.



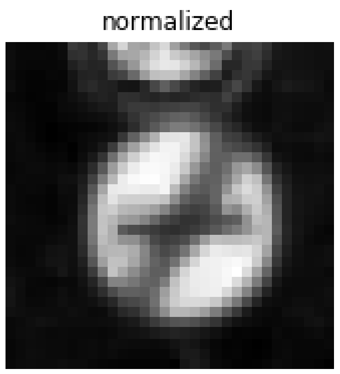
```
[ 180 1980 2010 1260 1770 1650  360 1290 1260 1320 1800 1170 1890 1920  690
  540  360  990 1080  180  300  270  330  450  240 1350  540  210  480  240
  390  690  210  599  360 1080  330  180 1860  270  300  210  210]
minimum samples for any label: 180
```

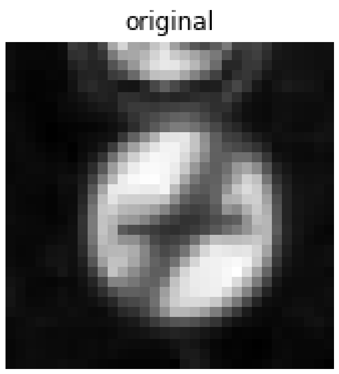
###Design and Test a Model Architecture

####1. Describe how you preprocessed the image data. What techniques 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. (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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

At first, I have converted the supplied training and test data to GrayScale. This is to ensure that I can cut down the expensive costs associated with processing large volumes of data and also under the assumption that the color is invariant in traffic sign classification. I have also normalized the grayscale images to ensure the mean of total training data is close to zero and equal variance. Below you can see the mean of training and test data before and after normalization.

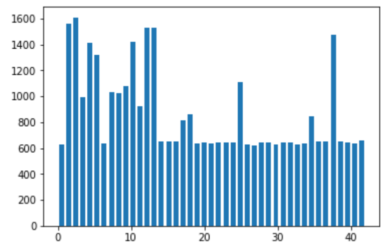
```
Mean of Training Data : 82.67758903699634
Mean of Test Data : 82.14846036120173
Mean of Normalized Training Data : -0.35408133564846583
Mean of Normalized Test Data : -0.3582151534281105
```





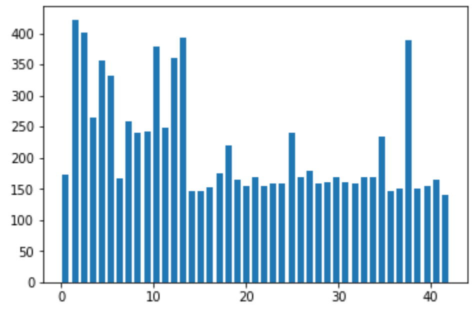
Next, I decided to discard the Udacity supplied validation dataset and create a new one from the training data as I felt that the given validation dataset may not be true representation of training set. But before carving out a new validation dataset, I decided to add additional input data to given training set in such a way that each class of input data has a minimum of 800 samples. This will ensure that the network when trained is not biased based on most occurring classes of inputs. Post data addition, I have split the new training data in such a way that 20% of it is the new validation set. See below visual that show the frequency of the new training and validation set by class/label.

Training Data Set – Post Data augmentation and split



```
[ 628 1558 1608  995 1413 1319  633 1031 1020 1077 1421  922 1529 1527  654
  654  648  815  861  636  646  632  645  642  641 1109  631  622  642  640
  631  640  641  631  632  846  654  649 1472  649  645  635  660]
minimum samples for any label: 622
```

Validation Data Set – Post Data augmentation and split



```
[172 422 402 265 357 331 167 259 240 243 379 248 361 393 146 146 152 175
 219 164 154 168 155 158 159 241 169 178 158 160 169 160 159 169 168 234
 146 151 388 151 155 165 140]
minimum samples for any label: 140
```

I have not modified the training data in any other way, though I have developed a way to modify and train the dataset. I have created a parameter called 'ROTMAT', which when adjusted by a value between 0 and 100, will modify a percentage of input image dataset (training dataset) in such a way that the image when visualized is rotated by 180 degrees. This is to ensure that Convolution network is better trained from position invariance perspective.

####2. Describe 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 final model consisted of the following layers: This is based on recommendations suggested by Sermanet and Yann.

Layer	**Description**
Input	32x32x1 Gray Scale image
Convolution 1	1x1 stride, Valid padding, outputs 28x28x6
Activation 1	RELU

Layer	Description
Max pooling 1	2x2 stride, outputs 14x14x6, Valid Padding
Convolution 2	1x1 stride, Valid padding, outputs 10x10x16
Activation 2	RELU
Max pooling 2	2x2 stride, outputs 5x5x16, Valid Padding
Convolution 3	1x1 stride, Valid padding, outputs 1x1x400
Activation 3	RELU
Flatten output of Max Pooling 2 – (Flatten A)	Output 400x1
Flatten output of Activation 3 – (Flatten B)	Outputs 400x1
Concatenate Flatten A and Flatten B	Outputs 800x1
Fully connected	Input 800x1; outputs 43x1

####3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyper parameters such as learning rate.

To train the model, I modified the network based on recommendations suggested by Sermanet and Yann. My network consisted of three convolution layers and one fully connected layer where one of the input to the fully connected layer is branches from the second convolution layer. I have then tried to play with different values of hyper parameters to see which one provides me the best validation accuracy. In my case, batch size of 128 and a learning rate of 0.001 provided me with a 100% accuracy during the 18th epoch.

####4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. 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.

My final model results were:

- training set accuracy of - 99.9%
- validation set accuracy of - 99
- test set accuracy of -

I first started developing on top of the original LeNET CoVnet model supplied by Udacity. I initially used the validation set provided by Udacity to test the training accuracy without modifying the input in any way. My initial input was RGB and data was not augmented or jittered in any way. With this, I was able to obtain a maximum training accuracy of 83%.

I then, tried to modify the hyper parameters by increasing and decreasing the learning rate. When decreased from an initial value of 0.001, the network seemed to train better but the accuracy still did not increase beyond 85%. And when learning rate increased above 0.001, the accuracy dropped below 80%.

I later adjusted the batch size up and down but felt retaining the batch size of 128 yielded better results.

I have then converted the input to grayscale and normalized it before training the network again. This time with a learning rate of 0.001 and batch size of 128, the accuracy achieved increased to 90%. However this was not sufficient as I wanted a better accuracy than that. Again I started playing with hyper parameters but nothing yielded better results.

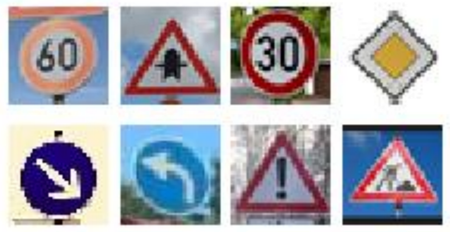
Later I made improvements to the network suggested by Sermanet and Yann. I have modified the network into a three stage Covent with one fully connected layer at the end. Input to fully connected layer is a branched output from the second Convolution layer which then combined with the output of the third Convolution layer. This model immediately improved the network accuracy by many folds. The accuracy achieved was 96%. Later I have introduced a drop out with a keep probability of 0.5(50%) during training and this increased the accuracy by about another 2% to 98.

Finally, I decided to add new data (again per recommendation by Sermanet and Yann) wherein I replicated the input for each class/label such a way that I have a minimum of 800 inputs for each class. This did magic and my accuracy increased to 99.9%

###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 few German traffic signs that I found on the web:



Some of these images are difficult to classify as they have additional edges than the ones we used to train the network. Especially, the last one (Road work) has extra edges on the top and bottom, making it difficult to classify.

####2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. 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).

Here are the results of the prediction:

Image	Prediction
Speed limit (60km/h)	Speed limit (60km/h)
Right-of-way at the next intersection	Right-of-way at the next intersection
Speed limit (30km/h)	Speed limit (30km/h)
Priority road	Priority road
Keep right	Keep right
Turn left ahead	Turn left ahead
General caution	General caution
Road work	Speed limit (30km/h)

The model was able to correctly guess 7 of the 8 traffic signs, which gives an accuracy of 87.5%. This compares little worse than the accuracy on the test set of 94.2%. I should have probably jittered the training input to improve the accuracy here.

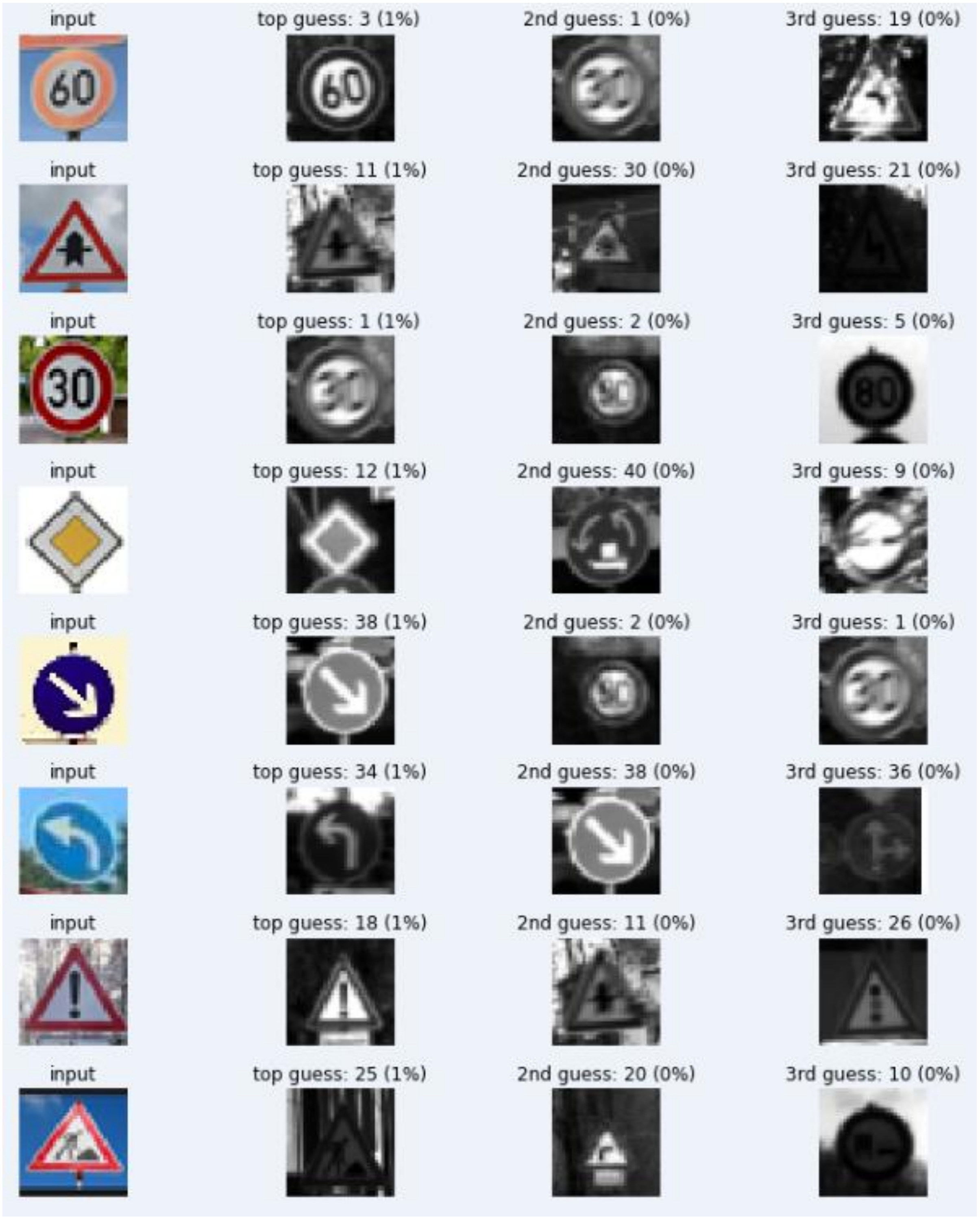
####3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions"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

Below are the top 5 probabilities for each input image.

```
Image 0 probabilities:
Speed limit (60km/h): 0.9923
Speed limit (30km/h): 0.0074
```

```
Dangerous curve to the left: 0.0002
Speed limit (80km/h): 0
Speed limit (50km/h): 0

Image 1 probabilities:
Right-of-way at the next intersection: 1
Beware of ice/snow: 00
Double curve: 0
Traffic signals: 0
Wild animals crossing: 0

Image 2 probabilities:
Speed limit (30km/h): 1
Speed limit (50km/h): 0
Speed limit (80km/h): 0
Speed limit (60km/h): 0
Speed limit (20km/h): 0

Image 3 probabilities:
Priority road: 1
Roundabout mandatory: 0
No passing: 0
No passing for vehicles over 3.5 metric tons: 0
Speed limit (50km/h): 0

Image 4 probabilities:
Keep right: 1
Speed limit (50km/h): 0
Speed limit (30km/h): 0
Speed limit (120km/h): 0
Roundabout mandatory: 0

Image 5 probabilities:
Turn left ahead: 1
Keep right: 0
Go straight or right: 0
Ahead only: 0
Yield: 0

Image 6 probabilities:
General caution: 1
Right-of-way at the next intersection: 0
Traffic signals: 0
Pedestrians: 0
Dangerous curve to the right: 0

Image 7 probabilities:
Road work: 0.9971
Dangerous curve to the right: 0.0025
No passing for vehicles over 3.5 metric tons: 0.0003
Go straight or left: 0
Double curve: 0
```

Below graphic shows the probability for each input image.



The code for making predictions on my final model is located in the 32nd cell of the Ipython notebook.

For the first image, the model predicted that this is a Speed limit (60km/h) (probability of 0.9923), and the image does contain a Speed limit (60km/h). The top five soft max probabilities are

Probability	Prediction
0.9923	`Speed limit (60km/h)`
0.0074	`Speed limit (30km/h)`
0.0002	`Dangerous curve to the left`
0	`Speed limit (80km/h)`
0	`Speed limit (50km/h)`

For the second image, the model is 100% sure that this is a "Right-of-way at the next int ersection" (probability of 1). The top five soft max probabilities are

Probability	Prediction
1	`Right-of-way at the next intersection`
0	`Beware of ice/snow`
0	`Double curve`

Probability	Prediction
0	`Traffic signals`
0	`Wild animals crossing`

For images 3 thru 7, the model is 100% sure on its predictions with probability of 1.

For the final image(#8) ... the model predicted with an accuracy of 99.71% . The top five soft max probabilities are

Probability	Prediction
0.9971	`Road work`
0.0025	`Dangerous curve to the right`
0.0003	`No passing for vehicles over 3.5 metric tons`
0	`Go straight or left`
0	`Double curve`