Traffic Sign Recognition

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

You're reading it! and here is a link to my (https://github.com/mlalaji07/CarND-Traffic-Sign-Classifier-Project/blob/master/Traffic_Sign_Classifier.ipynb)

###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 inbuild len() method to calculate summary statistics of the traffic

signs data set:

Number of training examples = 22271

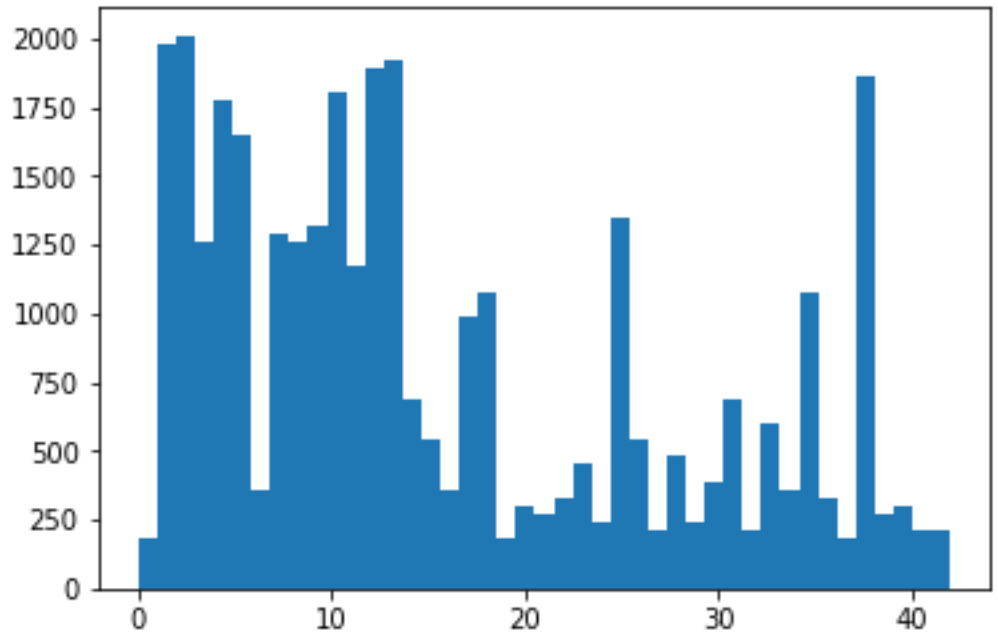
Number of validation examples = 4410

Number of testing examples = 12630

####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 IPython notebook.

Here is an exploratory visualization of the data set. It is a histogram showing the number of signs for each class.



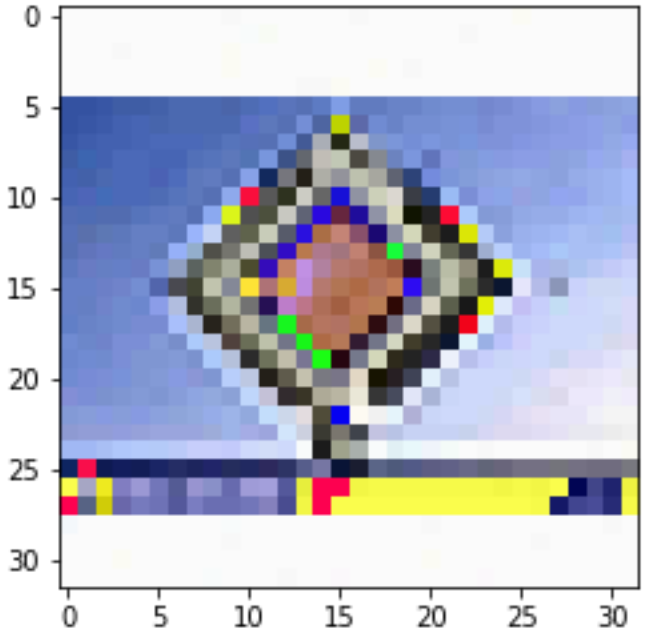
###Design and Test a Model Architecture

####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.

I did not convert to grayscale as I though converting to grayscale will mean losing information that will be information for classification in the later stages.

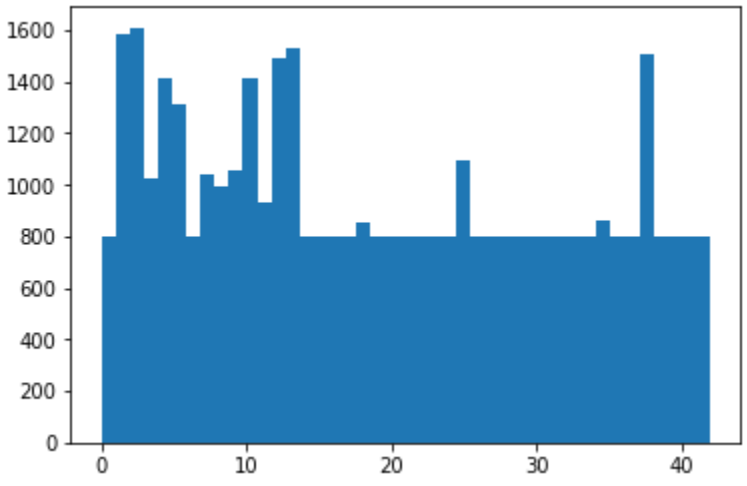
I normalized the image data in terms of mean and standard deviation.

Below is an example of the result after pre-processing. The original image was not included in the dataset. The original image was one of the pre-processed images.



I later saved the pre-processed images in a .pickle file so as to not repeat the process again.

I augmented my dataset by translating and rotating images. A threshold of 800 images per class was set wherein classes with images lower than the threshold were augment upto the threshold.



####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 seventh 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 split method.

```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
```

I have used mean substraction and normalization to pre-process the data.

-> Also, I am rotating the image and translating the image to augment the data. I am making sure that atleast 800 samples exist for each sign.

-> The resultant histogram after augmentation is displayed in the output zone below. -> Saving the pre-processed data in .pickle files so that I do not have to run pre-processing again and again.

####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.

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

My final model is the same as the LeNet except I am introducing Dropout before the 4^{th} fully connected layer. I experimented Dropout before the 4^{th} and the 5^{th} layer and found out it was more effective before the 4^{th}.

####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 in the ninth cell of the ipython notebook.

I've used the Adam Optimizer as used in the LeNet network.

Also, after tweaking the hyperparameters, I found out that I was getting the most effective validation percentage using the following values:

Learning_rate = 0.0004
mu = 0
Sigma = 0.15
EPOCHS = 75
BATCH_SIZE = 100
KEEP_PROB = 0.6(Dropout percentage for training)
KEEP_PROB_Validation = 1.0(Dropout percentage for validation)

####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 ninth cell of the Ipython notebook.

My final model results were:

* validation set accuracy of 98.7%

* test set accuracy of 94.3%

If an iterative approach was chosen:

I started with the original LeNet architecture. I then added regularization, performed pre-preprocessing, started tweaking the hyperparameters.

* Dropout and Learning rate were adjusted based on decrease in reduction of the validation accuracy at the later parts of validation..

If a well known architecture was chosen:

LeNet was chosen because is was tried on the MNIST architecture and gave nice results for the MNIST architecture. I think it does very well with images.

###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.

The images were chosen based on getting different type of signs, translations(position).

####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 eleventh cell of the Ipython notebook.

The images were difficult to classify because they are blurry, sharpening them may improve the accuracy of the trained classifier.

The model was able to correctly guess 1 of the 5 traffic signs, which gives an accuracy of 20%. I was also getting 40% in previous runs. As specified above, the primary problem associated is the blurriness of the images. Sharpening them would have made a huge difference.

####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 12th cell of the Ipython notebook.

Here is the output of the softmax function. Apart from the last image, all others point to just 1 image.

indices=array([[28, 0, 1, 2, 3],

[32, 0, 1, 2, 3],

[13, 0, 1, 2, 3],

[28, 0, 1, 2, 3],

[11, 31, 0, 1, 2]]))