#**Traffic Sign Recognition**
##Writeup Template

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

###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.
The project code is submitted as ZIP file along with this template.

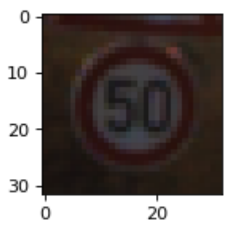
Source Code file created using Jupyter Notebook
Source Code file name: Traffic_Sign_Classifier.ipynb

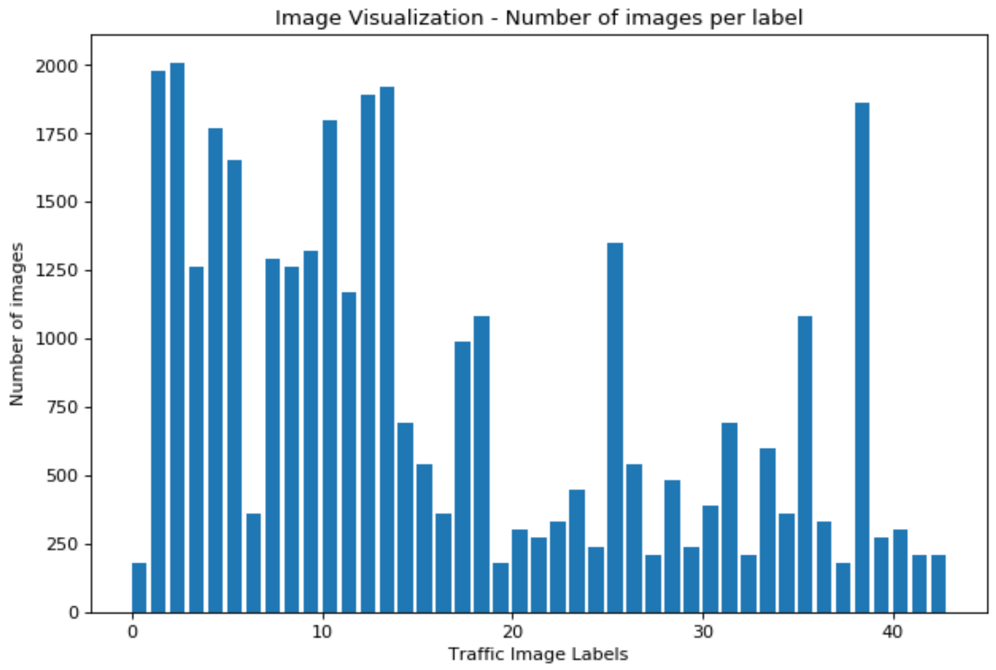
###Data Set Summary & Exploration
####1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.
The code for this step is contained in the second code cell of the

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

- 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. 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 ...





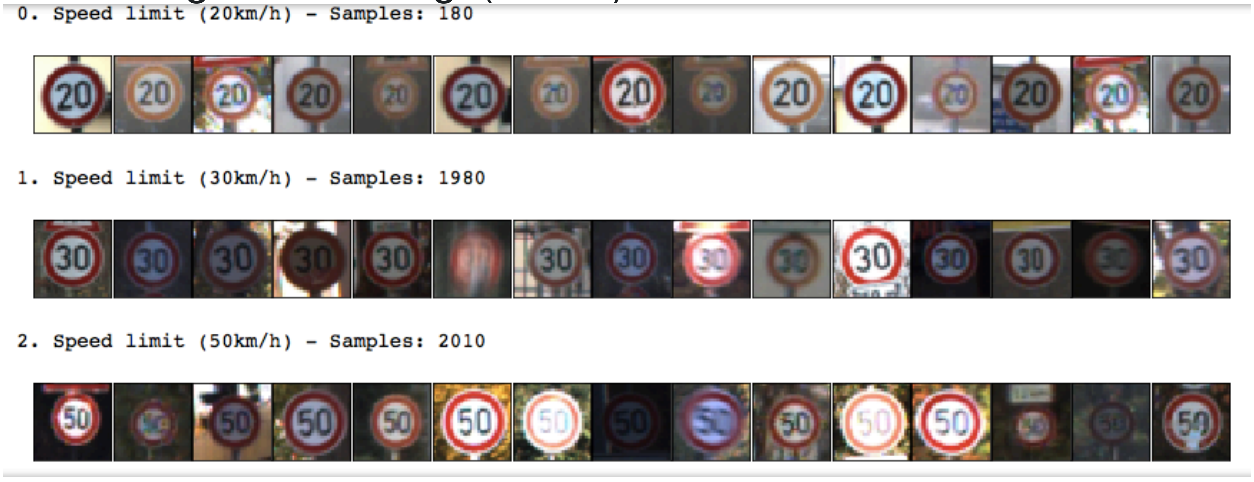
###Design and Test a Model Architecture
####1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques were chosen and why did you choose these techniques? Consider including

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 third and fourth code cell of the IPython notebook.
As a first step, I decided to print the mages by categorizing them using the image file names and the list of unique classes/labels provided in the signnames.csv file.

I am also using the COLOR_RGB2YCrCb to convert the image file to grayscale. Since it is common method ended up creating a common definition function PreprocessAnImages that takes the image as an input and returns the image by normalizing and converting the image to grayscale.

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

Initial Image Processing: (before)

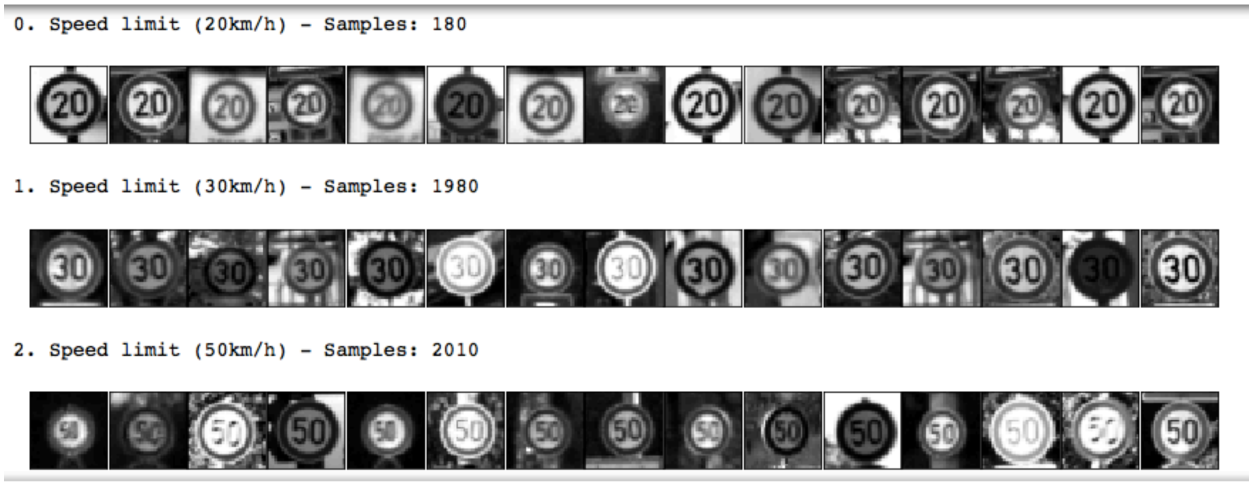


```
0. Speed limit (20km/h) - Samples: 180
```

```
1. Speed limit (30km/h) - Samples: 1980
```

```
2. Speed limit (50km/h) - Samples: 2010
```

Grayscaling of the images: (after)


0. Speed limit (20km/h) - Samples: 180
1. Speed limit (30km/h) - Samples: 1980
2. Speed limit (50km/h) - Samples: 2010

As a last step, I normalized the image data because normalization helps the model to not work harder to figure out the global minimum and/or determining the right values for the weight & bias.

####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.
Not Applicable as the Training, Testing & Validation data was provided as part of this project. Thus, we were not required to split the data.

####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 fifth cell of the ipython notebook under the section “**Model Architecture**” I ma using the code from the Solution: LeNet In TensorFlow given in the CNN module.

My final model consisted of the following layers:

Layer	**Description**
Input	32x32x1 RGB image
Convolution 3x3	1x1 stride, valid padding, outputs 28x28x48
RELU	
Max pooling	2x2 stride, outputs 14x14x28
Convolution 3x3	valid padding, outputs 10x10x96
RELU	
Max pooling	Outputs 5x5x96
Convolution 3x3	valid padding, outputs 3x3x172
RELU	
Max pooling	Outputs 2x2x172
Flatten	Outputs 688
Fully connected	Outputs 84
RELU	
Fully connected	Outputs 43

####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 fifth and sixth cell of the ipython notebook.
There are two placeholders for x and y where x is a placeholder for a batch of input images and y is a placeholder for the same batch of output labels that would be used in training the model.
In the same cell I calculate the Cross Entropy by calling the tensorflow function tf.nn.softmax_cross_entropy_with_logits which takes the logit as input parameter. Logit gets calculate using the LeNet. Using the reduce mean function on cross entropy the loss is calculated and that loss is the used by the AdamOptimizer to create the training operation. The learning rate is 0.001.

To train the model, I used an code from the CNN module that captures the Correct Prediction and Accuracy operation using tensorflow session. I am using the Epochs size of 35 & Batch size of 128 for training. I tried with the batch size of 64 but then bumped it up to 128 similarly I started with Epochs of 10 as shown in the CNN example then bumped it up to 16 and then doubled it and then bumped it just to see if I get any better or worst performance but did not notice any thus kept it at 35.

####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 sixth cell of the Ipython notebook.

My final model results were:

```
Training...
EPOCH 1 ... Validation Accuracy = 0.839
EPOCH 2 ... Validation Accuracy = 0.886
EPOCH 3 ... Validation Accuracy = 0.885
EPOCH 4 ... Validation Accuracy = 0.909
EPOCH 5 ... Validation Accuracy = 0.924
EPOCH 6 ... Validation Accuracy = 0.919
EPOCH 7 ... Validation Accuracy = 0.933
EPOCH 8 ... Validation Accuracy = 0.941
EPOCH 9 ... Validation Accuracy = 0.944
EPOCH 10 ... Validation Accuracy = 0.930
EPOCH 11 ... Validation Accuracy = 0.956
EPOCH 12 ... Validation Accuracy = 0.940
EPOCH 13 ... Validation Accuracy = 0.948
EPOCH 14 ... Validation Accuracy = 0.938
EPOCH 15 ... Validation Accuracy = 0.946
EPOCH 16 ... Validation Accuracy = 0.939
EPOCH 17 ... Validation Accuracy = 0.943
EPOCH 18 ... Validation Accuracy = 0.959
EPOCH 19 ... Validation Accuracy = 0.957
EPOCH 20 ... Validation Accuracy = 0.963
EPOCH 21 ... Validation Accuracy = 0.951
EPOCH 22 ... Validation Accuracy = 0.961
EPOCH 23 ... Validation Accuracy = 0.955
EPOCH 24 ... Validation Accuracy = 0.946
EPOCH 25 ... Validation Accuracy = 0.969
EPOCH 26 ... Validation Accuracy = 0.958
EPOCH 27 ... Validation Accuracy = 0.967
EPOCH 28 ... Validation Accuracy = 0.969
EPOCH 29 ... Validation Accuracy = 0.970
EPOCH 30 ... Validation Accuracy = 0.970
EPOCH 31 ... Validation Accuracy = 0.969
EPOCH 32 ... Validation Accuracy = 0.970
EPOCH 33 ... Validation Accuracy = 0.970
EPOCH 34 ... Validation Accuracy = 0.970
EPOCH 35 ... Validation Accuracy = 0.969
Model saved
Test Accuracy = 0.953
```

- training set accuracy of ?
- validation set accuracy of 0.839 to 0.970
- test set accuracy of 0.953

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen? Same architecture was tried.
- What were some problems with the initial architecture? N/A
- 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.- N/A
- Which parameters were tuned? How were they adjusted and why? – Explained above
- 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?

If a well known architecture was chosen:

- What architecture was chosen? The LeNet architecture taught in the CNN module was used.

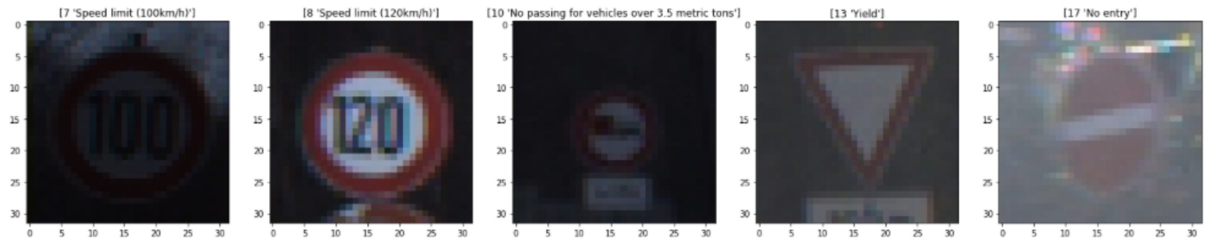
Why did you believe it would be relevant to the traffic sign application? LeNet was successfully created and trained for recognizing the handwritten digits. The Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

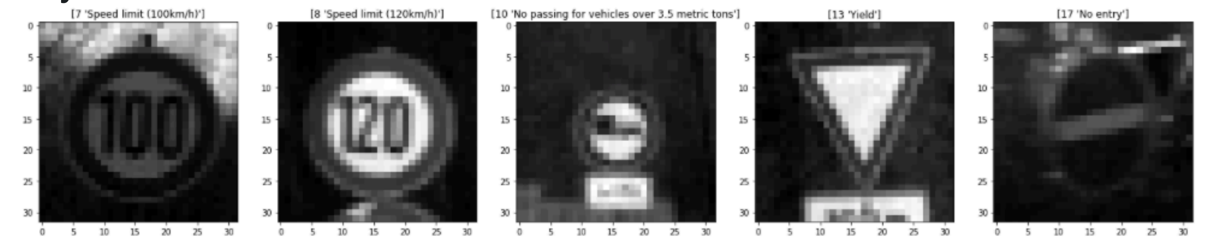
###Test a Model on New Images

####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:

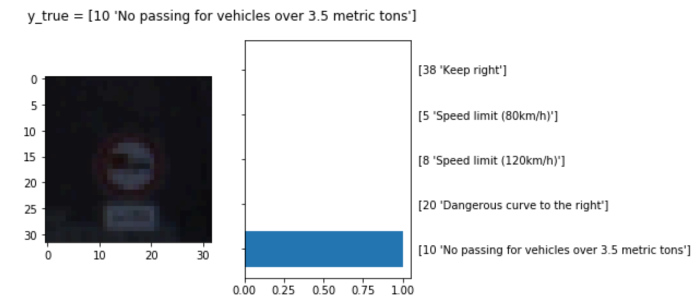

[7 'Speed limit (100km/h)']
[8 'Speed limit (120km/h)']
[10 'No passing for vehicles over 3.5 metric tons']
[13 'Yield']
[17 'No entry']

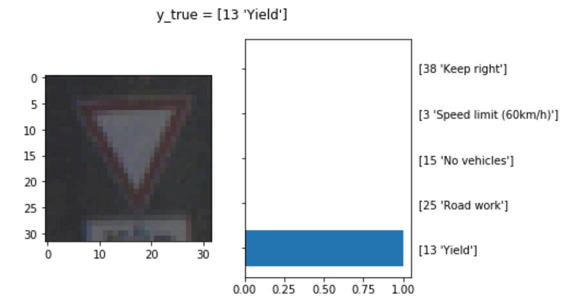
Grayscale Conversion

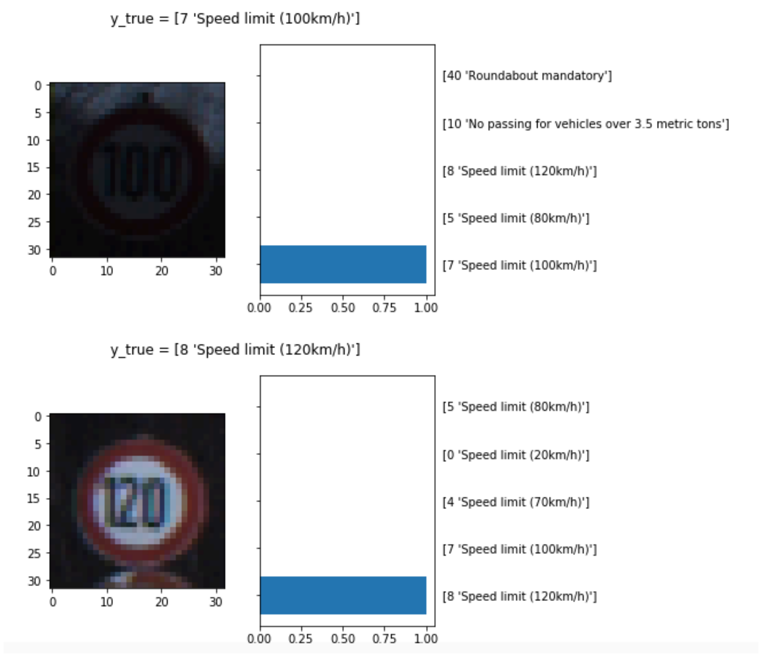

[7 'Speed limit (100km/h)']
[8 'Speed limit (120km/h)']
[10 'No passing for vehicles over 3.5 metric tons']
[13 'Yield']
[17 'No entry']

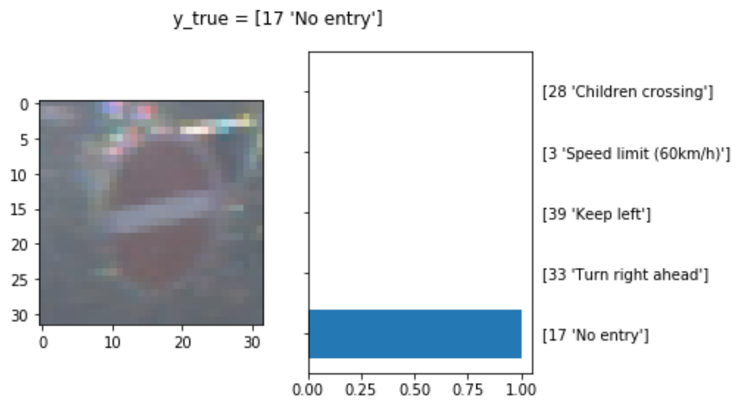
I Thought the first, third & fifth images might be difficult to classify because of the given darken and blurriness of the image.

####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 100	Speed Limit 100
Speed Limit 120	Speed Limit 120
No passing for vehicles over 3.5 metric tons	No passing for vehicles over 3.5 metric tons
Yield	Yield
No Entry	No Entry

The model was able to correctly guess 5 of the 5 traffic signs, wh ich gives an accuracy of 100%. This compares favorably to the a ccuracy on the test set calculated as: ... `Test Accuracy = 1.000`

####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)
N/A