**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**

**1.** You're reading it! The project code is submitted as a python notebook in the same archive as this writeup document.

**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.

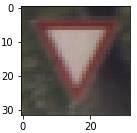
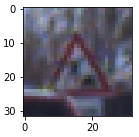
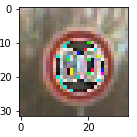
I used python to calculate summary statistics of the traffic signs data set:

* The size of the training set is 34799
* The size of the validation set is 4410
* The size of the test set is 12630
* The shape of a traffic sign image is (32, 32, 3)
  + The images are uncompressed RGB data.
* The number of unique classes/labels in the data set is 43

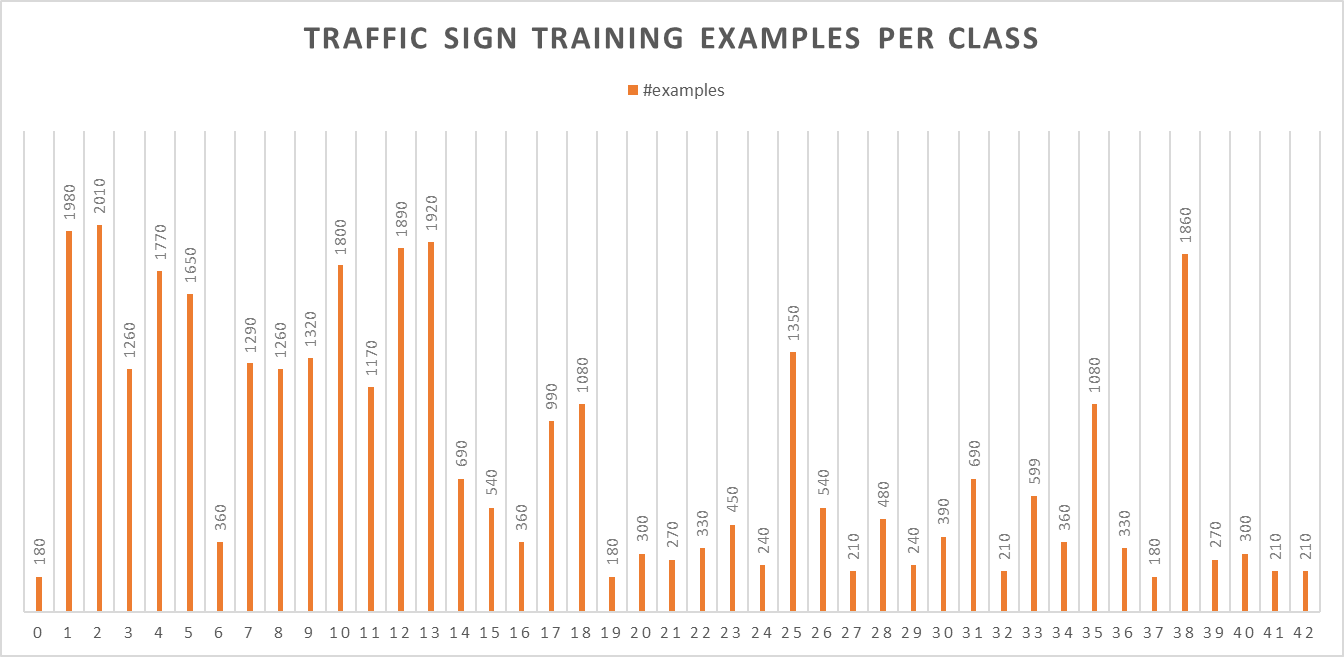
**2. Include an exploratory visualization of the dataset and identify where the code is in your code file.**

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

Here a few examples of images found in the training set:

Below is an exploratory visualization of the data set. It is a bar chart showing how many of each type of traffic sign images are included in the dataset. Note that in some cases there is significant deviation from the mean, e.g. class 0 - speed limit (20km/h), class 2 - speed limit (50km/h).



**Design and Test a Model Architecture**

**1. Describe how, and identify where in your code, 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.**

The code for this step is contained in the fourth and fifth code cells of the IPython notebook.

Other attempts to improve validation accuracy using data pre-processing had little effect and so were disabled (code was left for reference). These included:

* balancing the dataset by cloning images for signs with fewer numbers of examples
* grayscale conversion

As a last step, I normalized the RGB image data (range 0:255) to range -1:1 to improve network training performance. This had a noticeable positive impact on validation accuracy improving it by approximately 4-5%.

**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 training, validation and testing data are provided as separate inputs into the code so no provision for splitting the data is required.

**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 in the sixth cell of the ipython notebook.

My final model is a variation of LeNet architecture consisting of the following layers:

| **Layer** | **Description** |
| --- | --- |
| Input | 32x32x3 RGB image |
| Convolution 5x5 | 1x1 stride, VALID padding, outputs 28x28x12 |
| RELU |  |
| Max pooling | 2x2 stride, outputs 14x14x12 |
| Convolution 5x5 | 1x1 stride, VALID padding, outputs 10x10x24 |
| RELU |  |
| Max pooling | 2x2 stride, outputs 5x5x24 |
| Fully connected | Input 600, output 120 |
| RELU |  |
| Fully connected | Input 120, output 84 |
| RELU |  |
| Fully connected | Input 84, output 43 |
| Dropout |  |

**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 in the seventh cell of the ipython notebook.

To train the model, I used an AdamOptimizer with learning rate of 0.001, batch size of 256, and 10 epochs.

**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 in the eight cell of the Ipython notebook.

My final model results were:

* training set accuracy of 0.993
* validation set accuracy of 0.937
* test set accuracy of 0.914

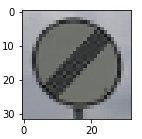
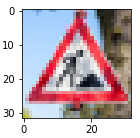
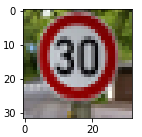
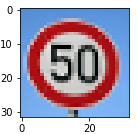
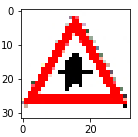
The architecture for the model was a slight variation of the LeNet architecture.

* LeNet was chosen because I was familiar with it and I suspected there are enough similarities in between traffic sign and MNist images to make it an effective classifier.
* Because the traffic sign images have more variation and general complexity than hand written numbers I decided to increase the depths of the convolution layers in hopes of extracting more underlying features. While this produced a small improvement in validation accuracy, it was less than I had hoped.
* Early validation runs suggested a tendency toward overfitting to the training data with training accuracy approximately 10% higher than validation accuracy. To compensate I added a dropout layer which helped to improve validation accuracy while consistently yielding training accuracies of 91-92%. I suspect that fewer epochs may produce comparable test accuracies but I opted to accept the results as sufficiently good.

**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:



I was unsure how these would classify as they were reproduced from jpeg compressed images. The third image has the most complex background and seemed most likely to be difficult to classify. The first image appears the most pixelated but it also has a clear triangular shape, a flat white background and a rather unique symbol.

**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 in the ninth cell of the Ipython notebook.

Here are the results of the prediction:

| **Image** | **Prediction** |
| --- | --- |
| Right-of-way | Right-of-way |
| Speed limit (50km/h) | Speed limit (50km/h) |
| Speed limit (30km/h) | Speed limit (30km/h) |
| Road work | Road work |
| End of all speed and passing limits | End of all speed and passing limits |

The model correctly predicted 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 91.4%.

**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 in the 10th cell of the Ipython notebook.

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

| **Probability** | **Prediction** |
| --- | --- |
| 1.00 | Right-of-way |
| 0.00 | Road narrows on the right |
| 0.00 | Double curve |
| 0.00 | Pedestrians |
| 0.00 | Speed limit (80 kh/h) |

The second image was correctly predicted to be a 50 km/h speed limit. The softmax probabilities were:

| **Probability** | **Prediction** |
| --- | --- |
| 0.9999 | Right-of-way |
| 0.00004 | Road narrows on the right |
| 0.0000007 | Double curve |
| 0.00 | Pedestrians |
| 0.00 | Speed limit (80 kh/h) |

Images 3 and 4 also were correctly predicted with similar very high softmax probabilities for the correct predictions.

Image 5, the “End Speed Limits” sign, was predicted correctly but with a probability of .955, the lowest of the 5 internet test images.

This outcome suggests that the test images from the internet may actually be contained in the training dataset.