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 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.

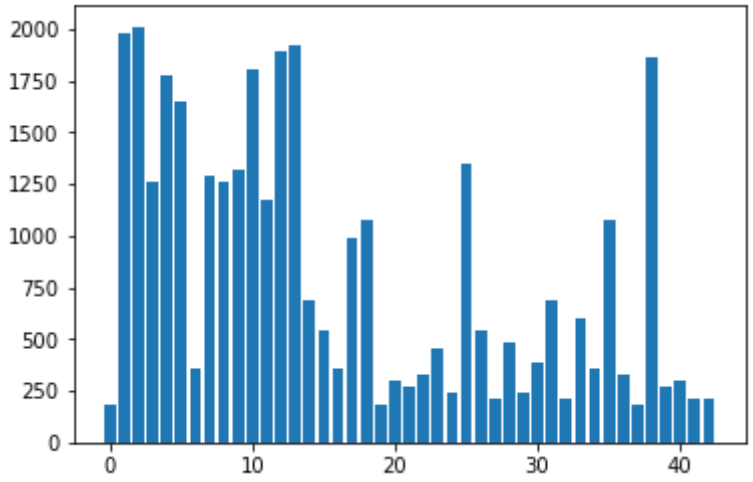
I used the numpy library to calculate summary statistics of the traffic signs data set:

- 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 fourth code cell of the IPython notebook.

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



###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 5th-8th code cell of the IPython notebook.

As a first step, I decided to show one image of each class because I wanted to decide how to preprocess the data.

Update:describes the preprocessing techniques used and why these techniques were chosen.

When looking at the image, I found a problem, some images were overexposed, and some were too dark to identify the signs. so I decided to normalize them.

I used skimage.exposure.is_low_contrast() to determine weather the image was overexposed or underexposed. Then, I used skimage.exposure.adjust_gamma() which can perform Gamma Correction on the input image to adjust overexposed or underexposed images.

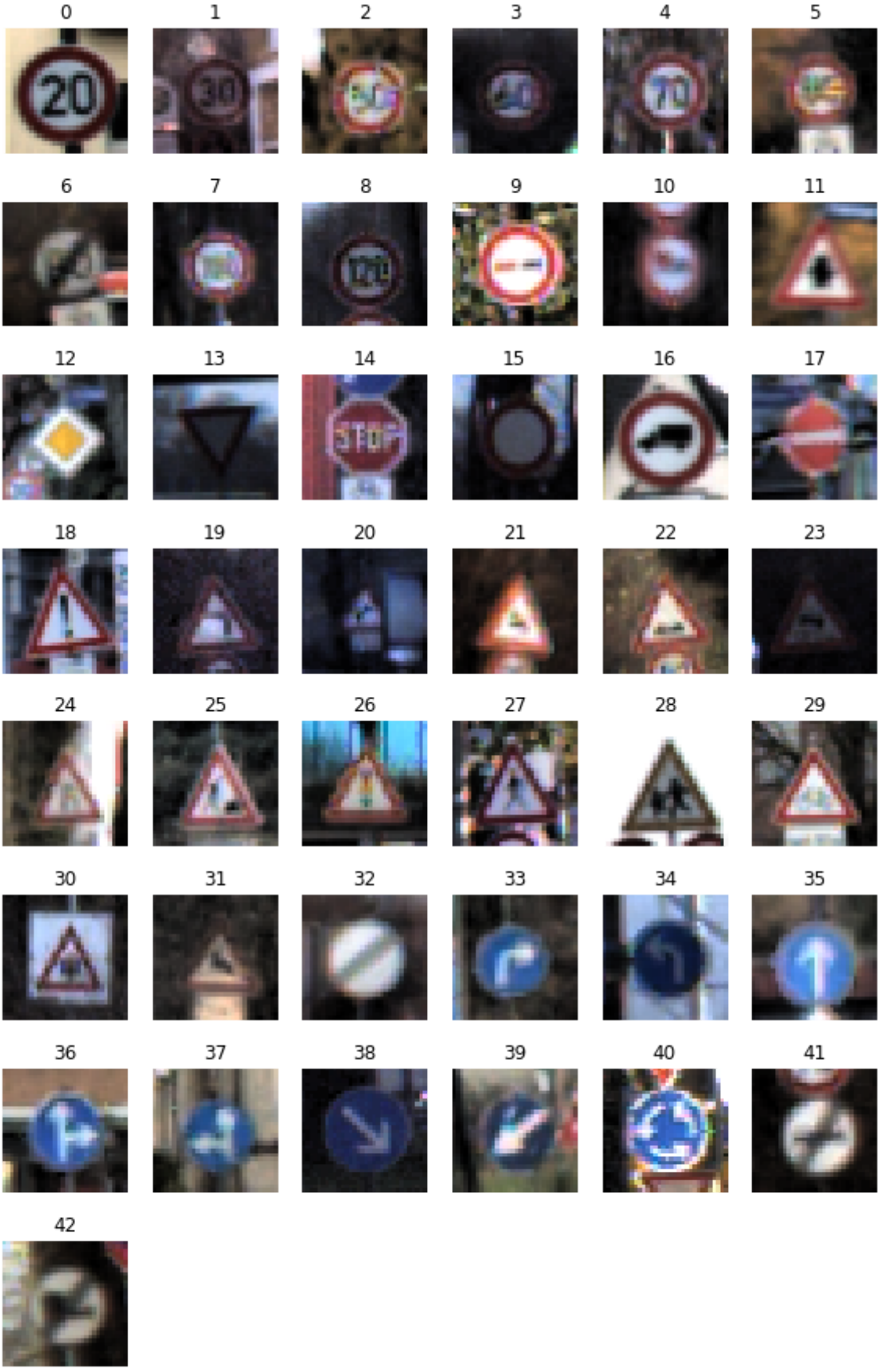
I also tested scikit-image.exposure.equalize_adapthist which can enhance local details based on Contrast Limited Adaptive Histogram Equalization (CLAHE) according to the documentation.

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

Before:

adjust_gamma:

equalize_adapthist:



Visually equalize_adapthist looks better.However, equalize_adapthist has precision loss when converting from float64 to uint16. And that made accuracy lower. All in all, adjust_gamma preprocess has a better accuracy.

####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)

Layer	Description

The code for generating additional data is contained in the 11th code cell of the IPython notebook.

I decided to generate additional data because the accuracy of first run was not good. To add more data to the the data set, I used cv2.getRotationMatrix2D because it's easy to rotate and rescale the images.

Here is an example of an original image and an augmented image:

The difference between the original data set and the augmented data set is the augmented date is rotated +-12° and rescaled 80% or 120%.

The size of final training set is 556784.

####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 14th cell of the ipython notebook.

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution 3x3	1x1 stride, VALID padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 3x3	1x1 stride, VALID padding, outputs 10x10x16
Flatten	outputs 400
Fully connected	outputs 120
RELU	
Dropout	avoid overfitting
Fully connected	outputs 84
RELU	
Dropout	avoid overfitting
Fully connected	outputs 43
softmax	

####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 16th-18th cell of the ipython notebook.

To train the model, I used parameters below:

- EPOCHS = 100 (for better accuracy)
- BATCH_SIZE = 1024 (for less oscillation)
- mu = 0
- sigma = 0.01
- rate = 0.001
- keep_prob: 0.9

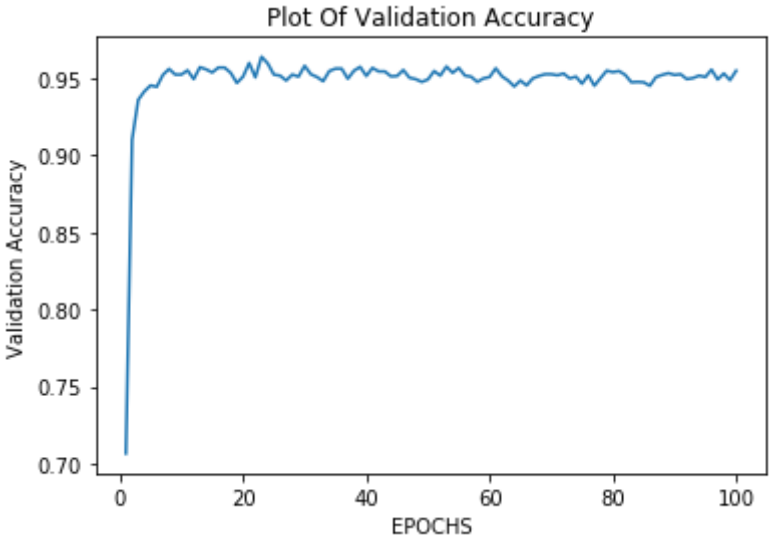
I also used sklearn.utils.shuffle to shuffle the data.

####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 20th cell of the Ipython notebook.

Here is a plot of training accuracy:



My final model results were:

- training set accuracy of 0.997
- validation set accuracy of 0.955
- test set accuracy of 0.939

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?

  Just simple LeNet. Because it works well with MNIST image.

- What were some problems with the initial architecture?

  At the begining, the accuracy is around 0.92. No matter how I changed parameters, the accuracy can not break through 0.93 stablely.

- 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.

  I figured out it was because of overfitting, so I add two Dropout layer with keep_prob 0.9. And It worked well.

- Which parameters were tuned? How were they adjusted and why?

  keep_prob. I have tried from 0.9 to 0.6 and the accuracy was decreasing so I picked 0.9.

- 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?

  Dropout layer can help avoiding overfitting and it worked 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:

The first image might be difficult to classify because it is partially blocked.

####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 22th cell of the Ipython notebook.

Here are the results of the prediction:

Image	Prediction
Vehicles over 3.5 metric tons prohibited	Vehicles over 3.5 metric tons prohibited
Priority road	Speed limit (80km/h)
General caution	General caution
Traffic signals	Traffic signals
No entry	No entry

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This is lower than the accuracy on the test set of 0.939.

####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 25th cell of the Ipython notebook.

For the first image, the model is completely sure that this is a Vehicles over 3.5 metric tons prohibited sign (probability of 1.00), and the image does contain a Vehicles over 3.5 metric tons prohibited sign. The top five soft max probabilities were

Probability	Prediction
1.00	Vehicles over 3.5 metric tons prohibited
.00	Speed limit (80km/h)
.00	Priority road
.00	Right-of-way at the next intersection
.00	Roundabout mandatory

For the second image, the model is completely unrecognized the Priority road sign. The top five soft max probabilities were

Probability	Prediction
0.45	Speed limit (80km/h)
0.16	Speed limit (70km/h)
0.12	Speed limit (50km/h)
0.09	No passing
0.05	No entry

For the third image, the model is completely sure that this is a General caution sign. The top five soft max probabilities were

Probability	Prediction
1.00	General caution
0.00	Pedestrians

Probability	Prediction
0.00	Traffic signals
0.00	Bumpy road
0.00	Speed limit (20km/h)

For the fourth image, the model is completely sure that this is a Traffic signals sign. The top five soft max probabilities were

Probability	Prediction
1.00	Traffic signals
0.00	General caution
0.00	Road work
0.00	Road narrows on the right
0.00	No vehicles

For the fifth image, the model is completely sure that this is a No entry sign. The top five soft max probabilities were

Probability	Prediction
1.00	No entry
0.00	Speed limit (20km/h)
0.00	Speed limit (30km/h)
0.00	Speed limit (50km/h)
0.00	Speed limit (60km/h)