Traffic Sign Recognition¶

Writeup ¶

Hong Cai

4/21/2017

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:

https://github.com/caihong1105/CarND-P2

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.¶

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

Total number of classes: 43

Number of training examples = 34799

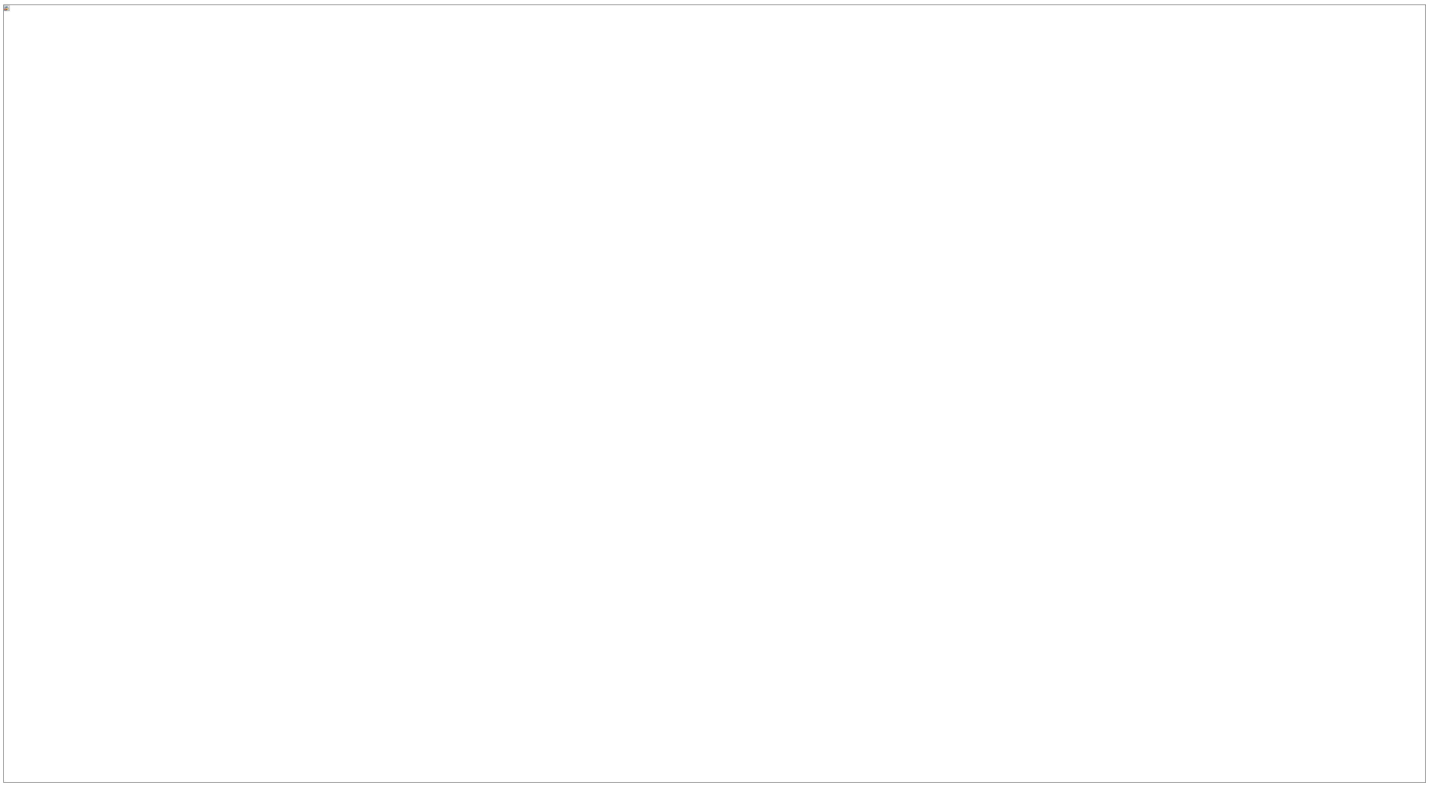
Number of testing examples = 12630

Image data shape = (32, 32, 3)

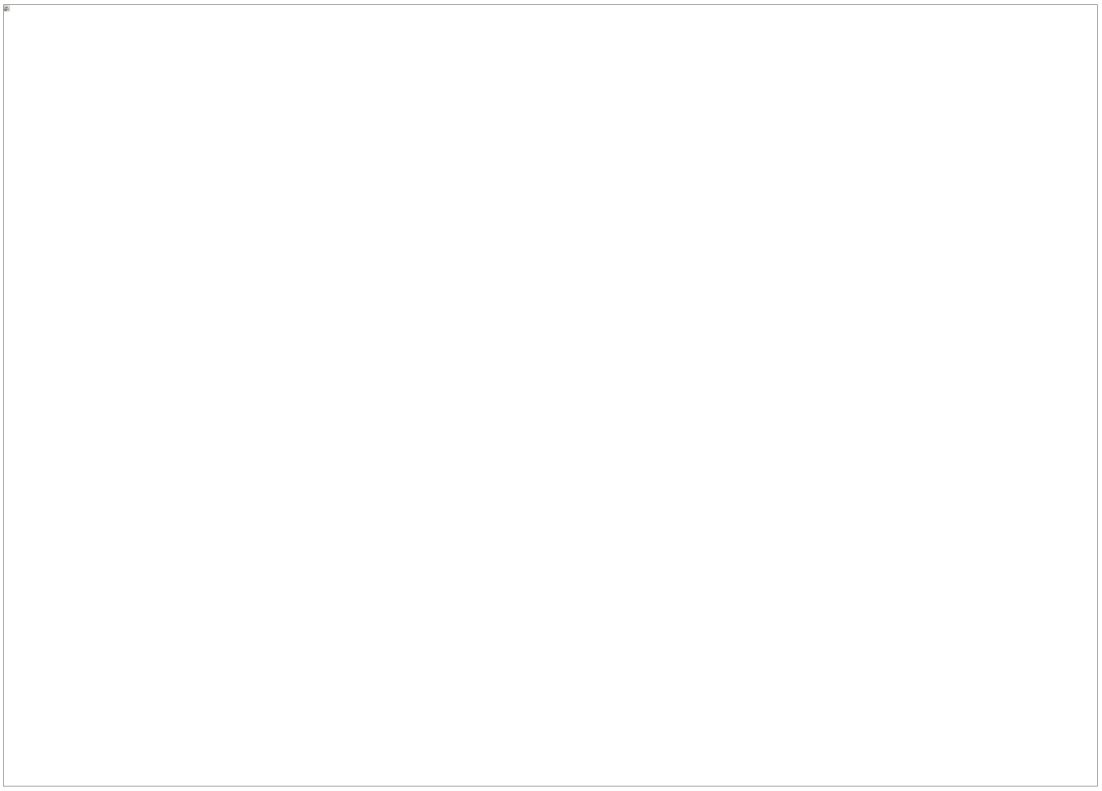
Number of classes = 43

2. Include an exploratory visualization of the dataset.¶

Here I visualize 20 examples picked randomly from train dataset.



Below are class distribution:



```
Highest count: 2010.0 (class 2)
Lowest count: 180.0 (class 0)
```

Below are the most common variations that I noticed when comparing the images in the dataset:

1. Variation of illumination (brightness/contrast).
2. Different filling factor, i.e the size of the sign versus the image size.
3. Variation of the background scenery and color.
4. Images from the same class can have very different histogram.

Also, in the same image there are similar variations. In particular, the brightness/contrast, the background are not uniform.

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.)¶

As a first step, I decided to convert the images to grayscale because at some references people say grayscale image maybe enough for taking key features.

But during my experiments I found that there could be other better image transformation method which I will describe below.

The data was pre-processed following the steps below:

1. the original dataset is splitted between the train set and the validation set in the ratio 4:1
2. Data augmentation.
3. Image conversion with below steps

   3.1 RGB2YCrCb

   3.1 cv2.equalizeHist(img_ycrcb[:,:,0])

   3.3 YCrCb2RGB

   3.4 prep = eqs_rgb/255. - 0.5 (normalization)

I also tried transformation into HUE domain etc. But found above transformation perform better.

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:

Layer	Description
Input	32x32x3 RGB image
Convolution 1: 3x3	1x1 stride, same padding, outputs 30x30x32
RELU	
Dropout	
Convolution 2: 3x3	1x1 stride, same padding, outputs 30x30x64
RELU	
Max pooling	2x2 stride, outputs 15x15x64
Dropout	
Convolution 3: 3x3	1x1 stride, same padding, outputs 12x12x128
RELU	
Dropout	
Convolution 4: 3x3	1x1 stride, same padding, outputs 12x12x256
RELU	
Max pooling	2x2 stride, outputs 6x6x256
Dropout	
Convolution 5: 3x3	1x1 stride, same padding, outputs 4x4x512
RELU	
Max pooling	2x2 stride, outputs 2x2x512
Dropout	
Flatten	outputs 2048
Fully connected	outputs 512
RELU	
Dropout	
Fully connected	outputs 128
RELU	
Dropout	
Softmax (final)	outputs 43

3. Describe how 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.¶

- **Optimizer: AdamOptimizer**
- **Batch size: 256 images** The batch size was varied from 64 to 500.
- **Sigma: 0.05** One of the parameters that affected the most the model performance was the standard deviation of the weight initializer. The weights values are initialized using a truncated Gaussian distribution with a mean of 0 and standard deviation *sigma*. If the standard deviation is too low or too high, the model will be barely learning during training. The loss would go down very slowly or not at all, and this independently of the learning rate chosen. In fact, depending on the normalization method used, the optimum sigma value may be different.
- **Learning rate: 0.001** I had to find the right balance between a learning rate too high where at worst the model does not learn, versus a learning rate too low where the convergence is slow. The learning rate is decreased to 0.0001 at EPOCH=20, and to 0.00005 at EPOCH=50, to prevent the system to be stucked in a local minima. Looks effective
- **EPOCHS: 100** I trained the model for 100 EPOCHS.
- **Dropout with keep rate of 0.5** The best keep rate was 0.5 for the fully connected layer. When the keep rate was below 0.5, the learning was at best slow with a loss that decreases very slowly. The convolutional layers used also dropout but with a higher *keep_rate* of 0.5.
- **Pooling** The pooling decreases the size of the output. It also prevents overfitting as a consequence of reducing number of parameters.
- **L2 regularization** Because the model still shows evidence of overfitting at high EPOCH (see Figure below), the L2 norm regularizer is applied to the weights of the Dense layers.

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:

- Train accuracy: 100%
- Validation accuracy: 99.89%
- Test accuracy: 98.84%

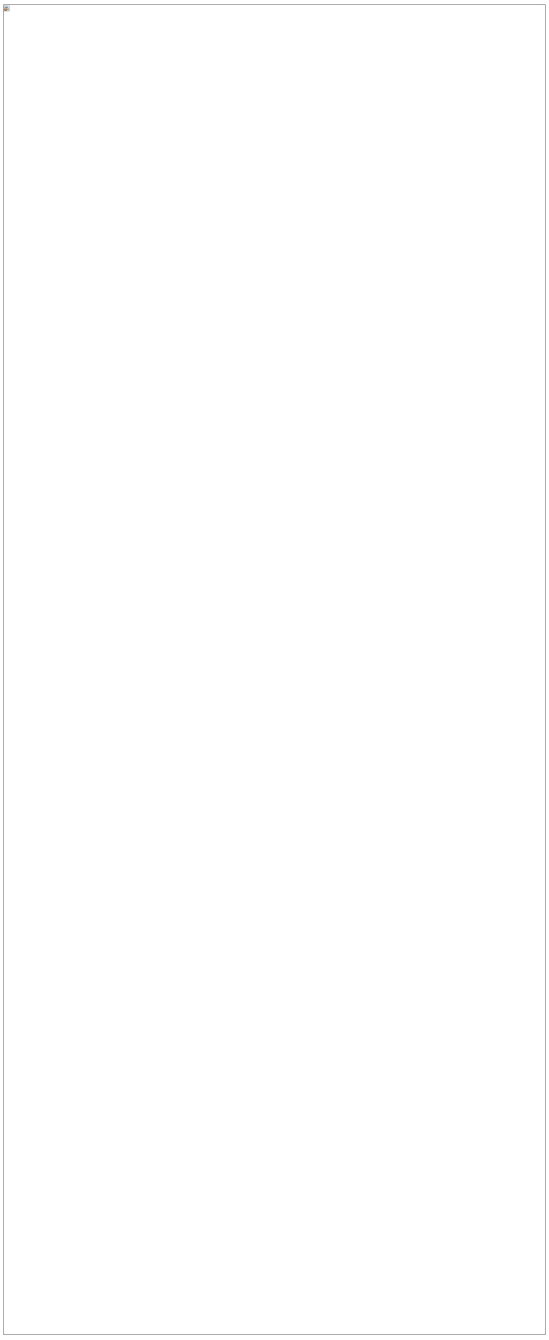
If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen? LeNet was first chosen because it's simple and is the first deep NN model taught in the class.
- What were some problems with the initial architecture? The accuracy is low (about 80%)
- 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 overfitting or underfitting. 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. Later I tried this one which is not necessarily so complex but effective for traffic sign data set.
- Which parameters were tuned? How were they adjusted and why? Almost all parameters were tried including "initialization parameters", "learning rate", "dropout rate", "L2 parameter", "batch size", "epoch number", same network architecture but different width (number of nodes in one layer).
- 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? It's a common sense now to use CNN for tasks of image recognition and classification. It's not an exception here for traffic sign classification. From experiments it seems that taking a dropout rate of 0.5 is a good starting point to prevent overfitting.

Test a Model on New Images¶

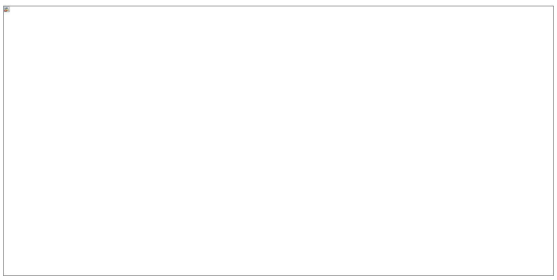
1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.¶

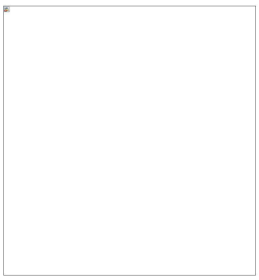
Here are ten German traffic signs that I found on the web together with their grayscale image.

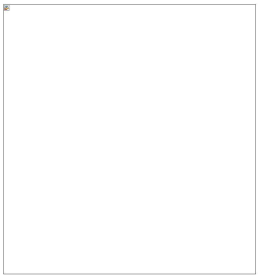


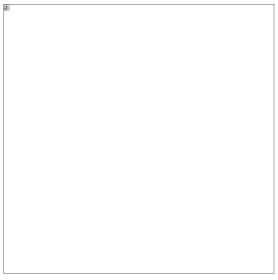
2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).¶

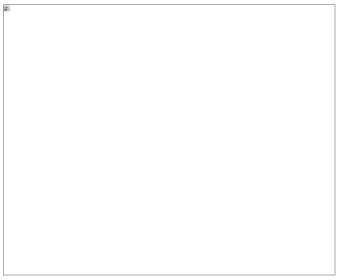
Here are the results of the prediction:

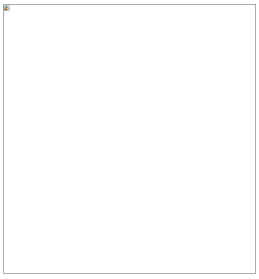


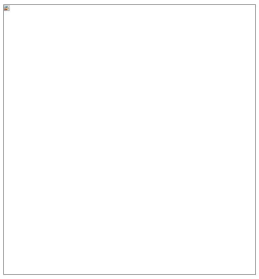


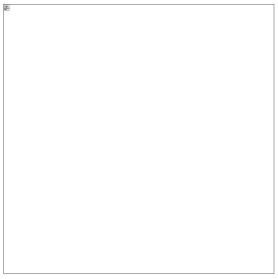












The model was able to correctly guess 5 out of 10 of new traffic signs. Much lower than test set.

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)¶