Carnd Traffic Sign Classifier

This is a traffic signs classifier written in TensorFlow.

Created through Udacity's Self-Driving Car Engineer program. It used German traffic signs to do ther learning.
This project is designed to follow these below steps:

1. Loading the dataset
2. Dataset summary, exploration and visualization
3. Design, train and test a model architecture
4. Use the model to make predictions on new images
5. Analyze the softmax probabilities of the new images

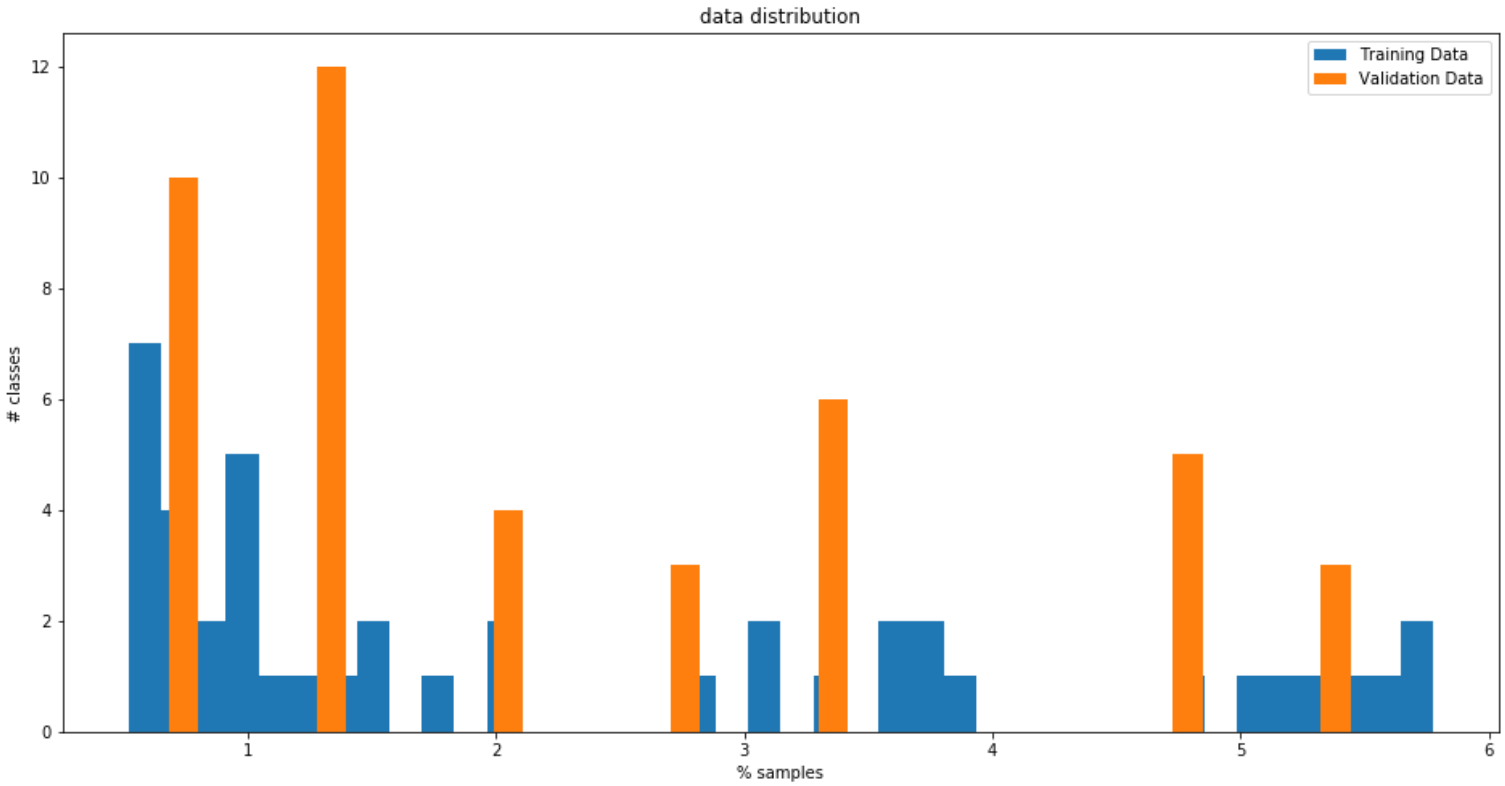
Loading the dataset

"Training dataset" came from :http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Downloads

Dataset summary, exploration and visualization

1. Number of training examples = 34799
2. Number of validation examples = 4410
3. Number of testing examples = 12630
4. Image data shape = (32, 32, 3)
5. Number of classes = 43

Here is the diagram of data set:



In the preprocessing we simplify the nomalization process for clearer and easier to process using :

```
image = cv.normalize(image,  image, 0, 255, cv.NORM_MINMAX)
```

The architecture we use for this project is a copy of classic LeNet.

1. convolution and ReLU with strides[1,1,1,1] padding=VALID # 32x32x3 => 28x28x25
2. maxpool with strides[1,2,2,1] ksize=[1, 2, 2, 1]padding=VALID # 28x28x25 => 14x14x25
3. convolution and ReLU with strides[1,1,1,1] padding=VALID # 14x14x25 => 10x10x40
4. maxpool with strides[1,2,2,1] ksize=[1, 2, 2, 1]padding=VALID # 10x10x40 => 5x5x40
5. convolution and ReLU with strides[1,1,1,1] padding=VALID # 5x5x40 => 3x3x60
6. flatP2 = flatten(C3) # 3x3x60 => 540
7. fully_connected # 540 => 120
8. drop layer at 50% # 120
9. fullyconnected # 120 => 43

Model Training:

Using Adam Optimizer for training at learning rate 0f 0.001. The batch size and number of epoch are 128 a nd 50.

Regarding about hyperparamenters, we have VALID padding, strides is [1,1,1,1] and ksize=[1, 2, 2, 1]

The accuracy on validation set is 0.936 (93.6%)

Test a Model on new images:

We use google to acquire five German sign. The performance on new image is 0.80 (80%)

Here is the **softmax probability** we obtain:

```
[[ 1.00000000e+00   1.83985366e-26   1.93113418e-38   0.00000000e+00
   0.00000000e+00]
 [ 2.25436658e-01   1.62351459e-01   1.57936841e-01   1.16493315e-01
   1.06655471e-01]
 [ 1.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00]
 [ 1.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00]
 [ 1.00000000e+00   6.00452433e-33   0.00000000e+00   0.00000000e+00
   0.00000000e+00]]
[[38 36 40  0  1]
 [30 25 38  1  5]
 [10  0  1  2  3]
 [38  0  1  2  3]
 [28 29  0  1  2]]
```