

### ###Data Set Summary & Exploration

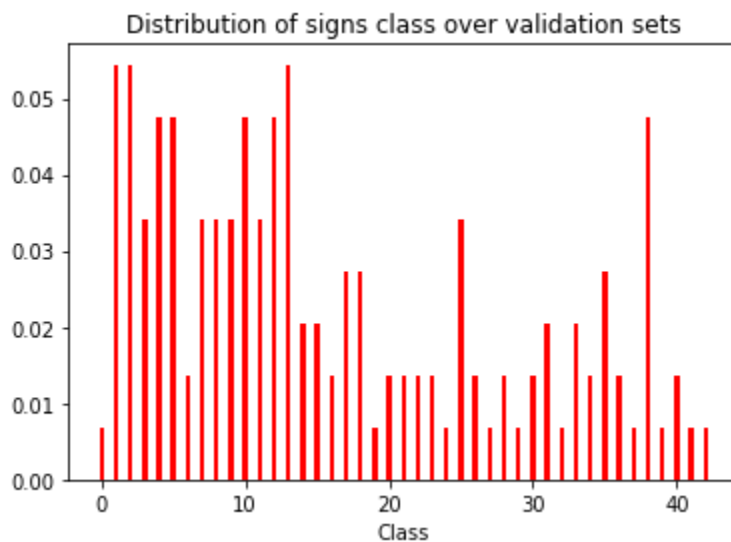
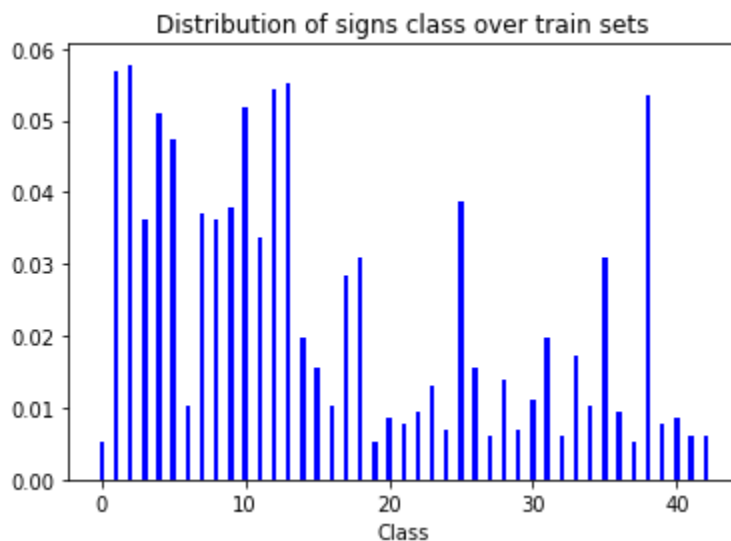
####1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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

- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is 32x 32 x 3
- The number of unique classes/labels in the data set is 43?

####2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set.



### ###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.

As a first step, I decided to convert the images to grayscale and normalize the image data so that the data has mean zero and equal variance

####2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity.

LeNet was chosen as the final model architecture. My final model consisted of the following layers:

**Convolution layer 1.** Convolution with 5x5 kernel, stride of 1, depth of 10, valid padding. The output shape should be  $28 \times 28 \times 10$ .

**Activation 1.** activation function `tf.nn.relu()`.

**Pooling layer 1.** Max pooling with 2x2 kernel, stride of 2, valid padding. The output shape should be  $14 \times 14 \times 10$ .

**Convolution layer 2.** Convolution with 5x5 kernel, stride of 1, depth of 16, valid padding. The output shape should be  $10 \times 10 \times 16$ .

**Activation 2.** activation function `tf.nn.relu()`.

**Pooling layer 2.** Max pooling with 2x2 kernel, stride of 2, valid padding. The output shape should be  $5 \times 5 \times 16$ .

**Flatten layer.** Flatten the output shape of the final pooling layer such that it's 1D instead of 3D.

**Fully connected layer 1.** This should have 120 outputs.

**Activation 3.** activation function `tf.nn.relu()`.

Drop out

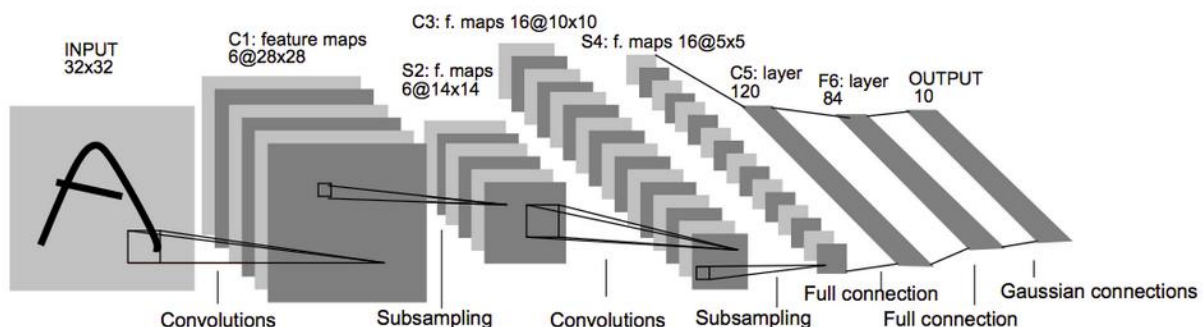
**Fully connected layer 2.** This should have 84 outputs.

**Activation 4.** activation function `tf.nn.relu()`.

Drop out

**Fully connected layer 3.** This should have 43 outputs.

The original LeNet-5 model looks like in the following chart.



Source: Yan LeCun

Optimizer is `tf.train.AdamOptimizer`

EPOCHS = 20

BATCH\_SIZE = 128

Learning rate = 0.001

Hyperparameters:  $\mu = 0$ ,  $\sigma = 0.1$ , drop out 0.99

My final model results were:

- training set accuracy of 0.998
- **validation set accuracy of 0.932**
- test set accuracy of 0.915

Validation set results:

- EPOCH 1 ...
- Validation Accuracy = 0.783
- 
- EPOCH 2 ...
- Validation Accuracy = 0.863
- 
- EPOCH 3 ...
- Validation Accuracy = 0.892
- 
- EPOCH 4 ...
- Validation Accuracy = 0.901
- 
- EPOCH 5 ...
- Validation Accuracy = 0.898
- 
- EPOCH 6 ...
- Validation Accuracy = 0.896
- 
- EPOCH 7 ...
- Validation Accuracy = 0.922
- 
- EPOCH 8 ...
- Validation Accuracy = 0.911
- 
- EPOCH 9 ...
- Validation Accuracy = 0.911
- 
- EPOCH 10 ...
- Validation Accuracy = 0.907
-

- EPOCH 11 ...
- Validation Accuracy = 0.930
- 
- EPOCH 12 ...
- Validation Accuracy = 0.911
- 
- EPOCH 13 ...
- Validation Accuracy = 0.928
- 
- EPOCH 14 ...
- Validation Accuracy = 0.927
- 
- EPOCH 15 ...
- Validation Accuracy = 0.916
- 
- EPOCH 16 ...
- Validation Accuracy = 0.916
- 
- EPOCH 17 ...
- Validation Accuracy = 0.926
- 
- EPOCH 18 ...
- Validation Accuracy = 0.921
- 
- EPOCH 19 ...
- **Validation Accuracy = 0.932**
- 
- EPOCH 20 ...
- Validation Accuracy = 0.930




If epochs is relative small value( for example, epochs < 10), A higher accuracy will be get if increase the epochs number. If epochs is a large value, the accuracy will be stable even increasing the epochs.



The high accuracy shows 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:

file	True label	Crop 32 x 32 x 3	difficulty
30.jpg	Speed limit 30km/h		easy
50.jpg	Speed limit 50km/h		moderate
Caution.jpg	General caution		Difficult  The words on the white plate under the general caution sign increase the difficulty

ahead_only.png	Ahead only		easy
no_entry.png	No entry		easy

####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.

The accuracy on these new predictions is 0.8. It is lower than the accuracy of the test set. The code has difficulty to classify the caution sign. The words on the white plate under the general caution sign increase the difficulty.

Here are the results of the prediction:

<b>Image</b>	<b>Prediction</b>
30km/h	30km/h
50km/h	50km/h
<b>Caution</b>	<b>Keep Right</b>
Ahead only	Ahead only
No Entry	No Entry

####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)

The code for making predictions on my final model is located in the 11th cell of the lpython notebook.