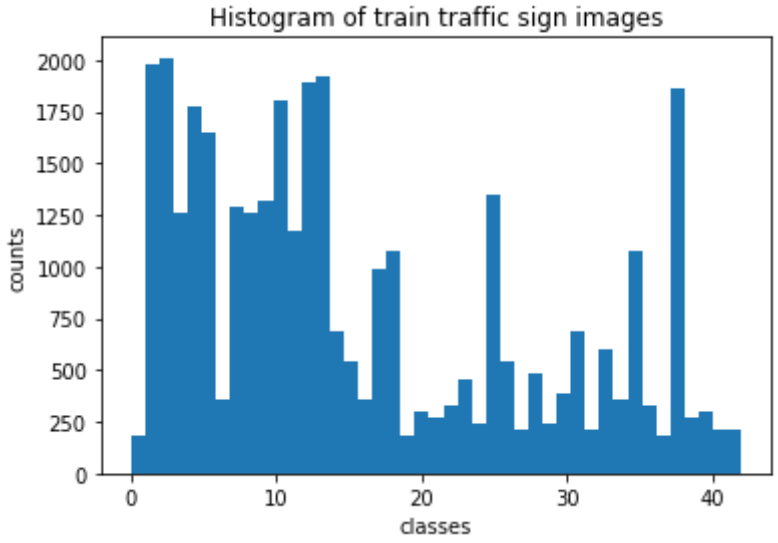
Traffic Sign Recognition

Build a Traffic Sign Recognition Project

Data set:
Number of training examples = 34799
Number of testing examples = 12630
Number of validation examples = 12630
Image data shape = (32, 32, 3)
Number of classes == 43

Visualization of dataset:

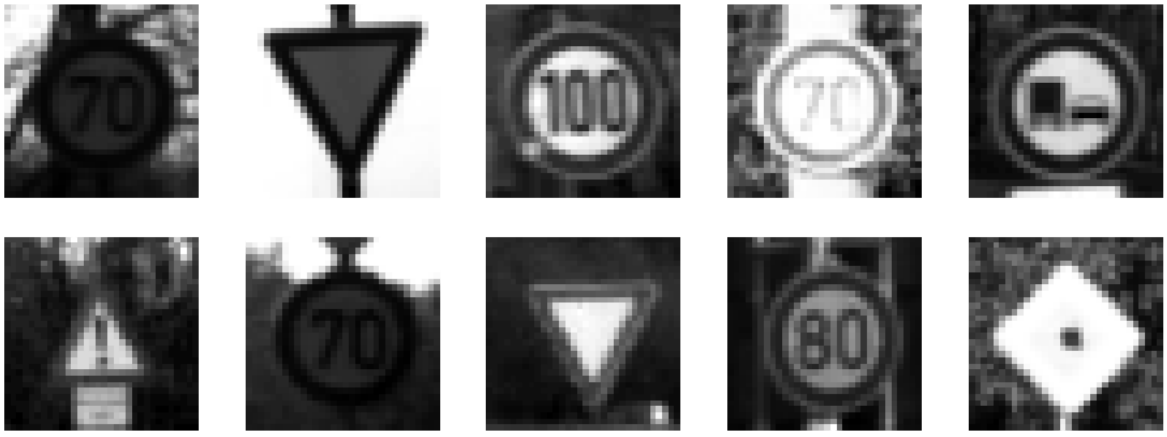




Preprocessing:
I tried to use
cv2.cvtColor(data[i], cv2.COLOR_RGB2GRAY) to convert rgb to gray

And normalize by the formula:
X_train_gray.astype(float) - 128) / 128

Here is the normalized gray images, we can see from the picture, they can be recognized by human eyes. But one channel is obvious easier than 3 channels to train a model. Because we reduce unnecessary noise.  Normalized is setting image range between [-1,1], also easier for module to process data.



Model Architecture:
Most of the part is from LeNet
layer1_shape (?, 28, 28, 6)
after maxpool layer1_shape (?, 14, 14, 6)
layer2_shape (?, 10, 10, 16)
after maxpool layer2_shape (?, 5, 5, 16)
flatten_layer_shape (?, 400)

layer_3 shape  (?, 240)
layer_4 shape  (?, 240)
logits shape  (?, 43)

But with connect two 400*400 fully connected layers to get a deeper network, and to avoid overfitting problem, I added a dropout after the fully connected layers, with keep_prob=0.4 when training:
layer_4 = tf.nn.dropout(layer_4, keep_prob)

Model Training:
Use:
EPOCHS = 20
BATCH_SIZE = 200
rate = 0.001

Questions:

- What was the first architecture that was tried and why was it chosen?
  My first architecture is original LeNet, which I got from last project exercise.
- What were some problems with the initial architecture?
  The different is original one has 10 outputs, need to change it as 43.
- How was the architecture adjusted and why was it adjusted?
  I change the last 3 layers as 400->240, 240->240, dropout 0.4 , 240->84.
  Because I tried to run the original LeNet, only got 0.903 validation and test accuracy but with 0.995 training accuracy. I thought I might overfit the training data. So I use dropout to mitigate overfit problem.
- Which parameters were tuned? How were they adjusted and why?
  See above.
- What are some of the important design choices and why were they chosen?
  I think most important part is using dropout to mitigate overfit problem. And increase the accuracy from 0.905 to 0.932

Solution Approach:
After I adjusted the network I got:
Validation Accuracy = 0.932
Test Accuracy = 0.932
Training Accuracy = 0.998

Acquiring New Images:
I searched 5 pictures from google, saved in download directories:
Normalized pictures are:

The brightness and contrast of picture I downloaded from google look good. But he orientation of the sign might be a problem. Like pic 2.

Performance on New Images
Test Accuracy = 0.600. Only got 3 of them are right. But the on the test set the accuracy is 0.93, I think I've kind of overfitted on training data.

On top of that, I found the speed limit sign is very easy to get wrong, since I tried different speed of signs, they all failed. I can see the number on the speed limit sign is hard to interpret by my model. Maybe I can improved by adding more data, or classify as speed limit sign first and train another model to predict its number, that should be more accurate.

Model Certainty - Softmax Probabilities:
indices=array([
[12, 14, 10, 40, 5],
[28, 3, 29, 1, 34],
[17, 34, 9, 8, 32],
[ 1, 2, 5, 7, 3],
[13, 25, 10, 5, 38]],
The original label should be
[12, 22, 17, 5, 13]
So we can see the second and fourth of pictures are predicted incorrectly.