This redo (in blue) based on the 1st Review

Nota Bene:

#cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits, one_hot_y)

cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=one_hot_y)

The above line was changed due to upgrading to Tensorflow 1.0 recently.

1) 1st Loading Data pickled Data

```
# Load pickled data
import pickle

# TODO: Fill this in based on where you saved the training and testing data

training_file = '../traffic-signs-data/train.p'
testing_file  = '../traffic-signs-data/test.p'

with open(training_file, mode='rb') as f: train = pickle.load(f)
with open(testing_file, mode='rb') as f:  test = pickle.load(f)

X_Otrain, y_train = train['features'], train['labels']
X_Otest, y_test   = test['features'], test['labels']
```

2) I then converted the data to gray scale

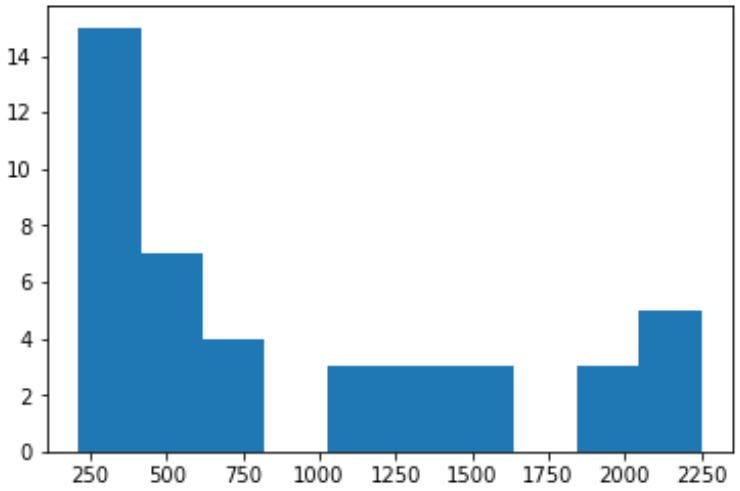
```
import numpy as np
import cv2

def cnv2gray(vec):
    return cv2.cvtColor(vec.astype(np.uint8), cv2.COLOR_BGR2GRAY)

X_Train = np.zeros([*X_Otrain.shape[:-1]])
for image in range(X_Otrain.shape[0]):
    X_Train[image] = cnv2gray(X_Otrain[image])
X_Test = np.zeros([*X_Otest.shape[:-1]])
for image in range(X_Otest.shape[0]):
    X_Test[image] = cnv2gray(X_Otest[image])
X_Train = np.expand_dims(X_Train, axis=4)
X_Test = np.expand_dims(X_Test, axis=4)
```

- The main reason for the gray scale conversion is because the model performed much better when converted, especially when I attempted to add my own image. The model performed horribly even with the better lighting condition and clearer image quality.
  - (see the last section)
- This also results in a faster compute as well, since gray scale images are less taxing with 1 vs 3 dimensions.
- Also may be issue is BGR vs RGB conversion confusion might have creep-ed in, as these images were found in various sources from the web.
- Overall, the ending results were much more robust.

3) The pickled data is a dictionary with 4 key/value pairs:
   - `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
   - `'labels'` is a 2D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
   - `'sizes'` is a list containing tuples, (width, height) representing the the original width and height the image.
   - `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**
4) Looking at the data
   - print(train['labels'][:20])
   - print(train['labels'][-20:])
   - `[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]`
     `[42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42]`
   - These are definitely order data
   - And the distribution of the data based on y Label shows that some signs occur far more than others.



   - I have split the data 80% train & 20% test as prescribed

5) The characteristics of the architecture & the type of model used.
   - Used conv2d model & maxpool in 2d within LeNet function with tensor flow as in lecture.
6) The number of layers in the model & the size of each layer.
   - Layers Consists of
     - CNN layer Input = 32x32x1. Output = 28x28x6.
     - Pooling Layer Output = 14x14x6.
     - CNN layer 2nd Output = 10x10x16.
     - Pooling Layer 2nd Output = 400.
     - Fully Connected1st Output = 120.
     - Fully Connected 2nd Output = 84
     - Fully Connected 3rd Output = n_classes or 43 different signs
     - Logit Layer

7) Please describe how you trained and evaluated your model and providing details on additional parameters like your batch size, learning rate, number of epochs and optimizer.
    - Fiddled around with different parameters, here are some attempts with significance
        - Tried: Gradient Descent 0.054
            - EPOCHS=100 BATCH_SIZE=62800
            - Test Accuracy 0.057
            - Definitly not way to go xxx
        - Tried: RMSPropOptimizer
            - EPOCHS=100 BATCH_SIZE=12800 .933
            - Test Accuracy 0.859
            - Better
        - Adams 0.306
            - EPOCHS=100 BATCH_SIZE=62800
            - Validation jumps around so I tuned SGD
            - Test Accuracy in 91% range
        - As shown above tried different batch & epoch for now. Will revisit
        - Seems like, it's hard to improve on the original model beyond Using the grayscale

8) Please discuss how did you choose the optimizer.
    - Just followed lecture & tried different combination as seen above.
    - RMS Prop Optimizer above was okay to start with, and when I decreased the learning rate the Validation Accuracy jumped around less & Smaller BATCH_SIZE seems to work better
    - At the end Adams seems to fair better as in test in 91% range

9) Please discuss how did you tell a convolutional layer is well suited for this problem. (Missed Answer)
    - Obviously, getting anything above 95% accuracy in any condition is a good indicator of the power of CNNs

10) Please discuss how did you choose the particular activation (Missed Answer)
    - Softmax was recommended & used throughout the lecture. Relu or Tanh did NOT seems to improve the answer. Conceptually, this made sense too. With 43 categories to choose from either high reward of correct response nor binary reward does not make sense. The logit at the end of the model also goes well with the curvature of Softmax.

11) Please discuss how did you tune the hyperparameter. (Missed Answer)
    - The hyperparameters were chosen with largely based on lectures plus trial and error. Increasing the batch size was rather troubling as it sped up the training and fit the training data better but did horribly at test data. Similar problems occurred when I attempted to increase the epochs. Except after a certain point (around 30 or so), epochs increase really did not help. (Below are some data)
    - #=EPOCH 10 ... Validation Accuracy = 0.933 BATCH_SIZE 128
    - #=EPOCH 60 ... Validation Accuracy = 0.948 BATCH_SIZE 1000

- #=EPOCH 100 ... Validation Accuracy = 0.971 BATCH_SIZE 128
  'does not improve beyond .98 after 30~ or so
- #=EPOCH 10 ... Validation Accuracy = 0.075 BATCH_SIZE 12800
- #=EPOCH 30 ... Validation Accuracy = 0.929 BATCH_SIZE 1280
- #=EPOCH 30 ... Validation Accuracy = 0.903 BATCH_SIZE 12 # slow & UNSTABLE!
- #=EPOCH 30 ... Validation Accuracy = 0.948 BATCH_SIZE 328 #starts low flatten around 25~
- #=EPOCH 30 ... Validation Accuracy = 0.960 BATCH_SIZE 128 # a winner with ***stablility***!!

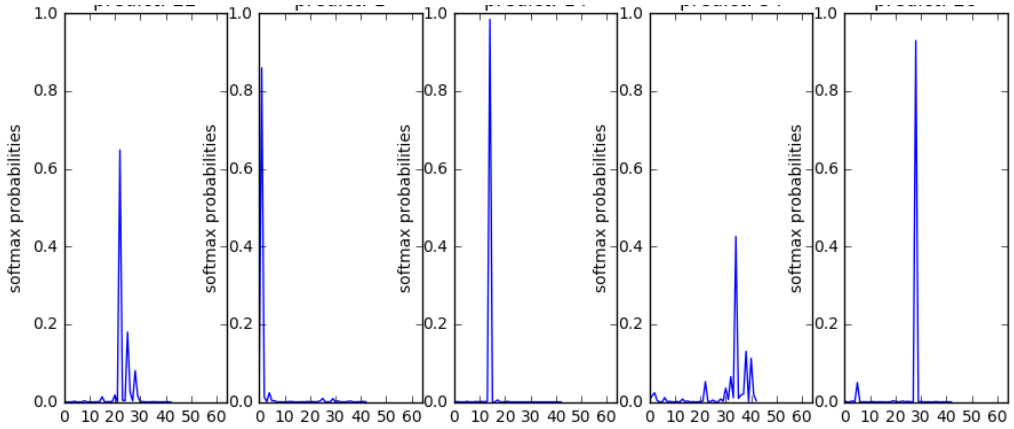
12) Please discuss as to any particular qualities of the images or traffic signs in the images that may be of interest, such as whether they would be difficult for the model to classify.

- The surprise was colors really did NOT help to identify the signs any better; however, at the end of this term nears, now we have been taught tools like how to use additional color separation, camera calibration, aspect correction, and HOG, these techniques may help to better the results. This would be in future to do list.
- Also issue is similar to vehicle finding. One needs to be able to identify the window with sign & not signs even when it's partially covered by other objects. That would layer an additional difficulties, especially in a moving car.
- So, this would be a challenging problem in the real world.

13) Please provide a discussion on the certainty and uncertainty of your models predictions and also some visualizations of the softmax probabilities of the predictions

Suggestion

•You might want to expand more on this section by providing a graph which can look a bit like this :



•

•*Having difficulties with this section. Could u pls instruct on how to grab (even for future ref). Will update the write up afterward.*

•Again using the gray scale transformation and the trained model I performed prediction on my supplied images.

•**Nota Bene** The last one is 130kmh sign,which has no classifier. I wanted to know would happen to the model with that. So, with that out it's prbably a bit higher.

with tf.Session() as sess:

saver.restore(sess, tf.train.latest_checkpoint('.'))

ntest_accuracy = evaluate(X_ntest, y_ntest)

print("New Test Accuracy = {:.3f}".format(ntest_accuracy))

•New Test Accuracy = 0.750

•Actually Accuracy is much higher, as last image was a fluke 130km is NOT in the classifier. Only 2/15 images wrong = 87% correct!!!!

•Also, this jumps around quite a bit. From 70~near 90%, when I rerun it.

•The below shows why. The mean & std of the images are completely off! We need to change the Mu & Sigma from above. This is somekind of color prblem, then this would not be corrected by simple mu, sigma correction however. And looking at the pics in matplot shows that issue. Only simple way to correct that would put the images in same color scale or gray scale the image

- Z scale failed miserably - see Traffic_Sign_Classifier-zScore.ipynb That's a dead End
- Obviously Gray Scale worked!!!!!!!

```
for i in range(y_ntest.shape[0]):
    ys = np.equal( y_test, y_ntest[i] )
    xs = X_Test[ys, :, :, :]
    xf = xs.flatten().reshape(xs.shape[0], (32*32*1))
    print ( 'O %3d %3d %6.2f %6.2f' %(max(xf.flatten()),
min(xf.flatten()), \

np.mean(xf.flatten()),np.std(xf.flatten())) )
    xn = X_ntest[i]
    print ( 'N %3d %3d %6.2f %6.2f' %(max(xn.flatten()),
min(xn.flatten()), \

np.mean(xn.flatten()),np.std(xn.flatten())) )
```

•O 254 12 69.23 60.33

•N 255 44 148.82 50.27

•O 255   9 76.74 63.79

•N 241   1 132.64 59.22

•O 255 18 86.38 54.87

•… (see notebook for rest)

• As another way to help with the certainty, it is by looking at the visualizations of softmax probabilities of a set of top K for an image, If there is a class with a way higher probabilities than other in its top K's for a particular image, that means your model was quite certain of this prediction and if the probabilities are nearly the same and not really distinguished, then the model was uncertain of that prediction.

•**Answer:**

```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    sess = tf.get_default_session()
    top3 = sess.run(tf.nn.top_k(tf.constant(X_ntest), k=1))
```

• ***Please see notebook for the results***

•print(top3[1][1][2].shape)

•x = "will give you class prediction probability for i-th data set, i_label is the index that goes from 0 to 3 corresponding probability scores of classes with highest to lowest chances of being the class in the test image."