Traffic Sign Classification Writeup:

Dataset Exploration

1) Dataset Summary

There were 2 separate datasets. The first download consisted of train.p and test.p. This dataset was then updated to train.p/test.p/valid.p. These datasets are not the same.

The images are stored as RGB vs BGR in the .p files.

2) Exploratory Visualization

Exploratory Visualization of dataset statistics:
Created histograms for valid.p/test.p/train.p showing the number of examples/class. There are varying samples per class. The ratios are consistent for the 3 files.

Design and Test a Model Architecture

Preprocessing:

There are 2 options; feed the complete 32x32 scaled images as provided into the model or do a processed grayscale of 32x32x1 with added rotated/translated/brightness adjusted images.

The first iteration was to do a grayscale conversion as was done with MNIST.

There were several points which took time:

1) the images come out as float64 from openCV, they have to be converted to float32 to match the TF variable float32 description for placeholder variables x and y. This code was copied from the LeNet examples in lecture.
2) The image normaliztion to +/- 1 has some values which are 0. 0 does not multiple well. This may cause an accuracy problem. The code examples in lecture later as in the behavorial cloning adjust to a positive range. This makes the adjustment to a 0 mean impossible though.
3) The image brightness adjustment is done using np.histogram. There are 2 steps for quality control for this;
    1) show the before and after histogram equalized images in the workbook. Can see the equalized images are brighter and the edges are more distinct.

2) plot the histogram. Setting the default=true/false moves the peak close or away from y=0. Unknown if either one really makes a difference or not. They both look the same on image plot.

4) Rotating images causes these black artifacts on the edge. I show this on an image plot. These black edges can be removed by doing a crop then an upscaling as shown in the test images. Tested but unknown if this makes a consistent difference. There is a problem with down sampling a 3 from a higher resolution to a lower resolution in the middle finger of the 3 is removed.

- ndimage does not work

5) the input into the NN is X, 32,32,1 where X is the number of images. Use newaxis to convert an image from [32,32] to [32,32,1]. Note that once you format with this method you cannot display these for inspection in an image plot.

Model Architecture:

LeNet:

Copied from lecture notes and added dropout from MNIST TF tutorial. This is different than the DO implemented in the Keras architecture.

Keras implemenation of LeNet w//dropout for 98% accuracy:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 32, 32, 32)        2432
_________________________________________________________________
relu1 (Activation)           (None, 32, 32, 32)        0
_________________________________________________________________
maxpool1 (MaxPooling2D)      (None, 31, 31, 32)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 31, 31, 64)        51264
_________________________________________________________________
relu2 (Activation)           (None, 31, 31, 64)        0
_________________________________________________________________
maxpool2 (MaxPooling2D)      (None, 15, 15, 64)        0
_________________________________________________________________
conv3 (Conv2D)               (None, 15, 15, 128)       204928
_________________________________________________________________
relu3 (Activation)           (None, 15, 15, 128)       0
_________________________________________________________________
maxpool3 (MaxPooling2D)      (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout1 (Dropout)           (None, 6272)              0
_________________________________________________________________
hidden1 (Dense)              (None, 128)               802944
_________________________________________________________________
```

```
relu4 (Activation)         (None, 128)            0
_________________________________________________________________
dropout2 (Dropout)         (None, 128)            0
_________________________________________________________________
hidden2 (Dense)            (None, 128)            16512
_________________________________________________________________
relu5 (Activation)         (None, 128)            0
_________________________________________________________________
output (Dense)             (None, 43)             5547
_________________________________________________________________
softmax (Activation)       (None, 43)             0
=================================================================
Total params: 1,083,627.0
Trainable params: 1,083,627.0
```

Model Training:

There are 2 models. The first one is implemented in TF and copies the lecture format. This is presented for ease of grading and to serve as a reference for future work on how to use TF. The training is stored as 10/30/100 Epochs. 200 Epochs was run on a Titan X and the place where the loss curve increases is at approximately 100 Epochs.

The second model is implemented in Keras. It turns out the dropout loss makes a difference. I copied the DO loss from the MNIST TF tutorial for the first model. For the second model the DO is moved to before the final layer.

Also in the second model the only normalization which was implemented was range normalization. This was at 98% accuracy so I stopped here since I am way past the deadline already.

Solution Approach:

The first step was to run LeNet as the default with conversion to greyscale, normalization to -/+ 1 and conversion to mean 0 as dictated in the lecture. Without adding data the accuracy was 93.% which meets the rubric criteria. This is stored as a cell in the workbook.

Test a Model on New Images

1. Acquiring New Images

Downloaded 5 images from internet and scaled down.

2. Performance on New Images

3/5 mismatch; 2/5 match.

3. Model Certainty - Softmax Probabilities

```
prob: 0.978146  class: 0 labels: 0
     prob: 0.0109876  class: 1 labels: 16
     prob: 0.00767156  class: 4 labels: 16
     prob: 0.00146429  class: 8 labels: 16
     prob: 0.000890397  class: 18 labels: 16
Mismatch.. (0, 1)
prob: 0.995239  class: 5 labels: 0
     prob: 0.00228423  class: 1 labels: 1
     prob: 0.00165667  class: 0 labels: 1
     prob: 0.000819964  class: 3 labels: 1
     prob: 1.35885e-07  class: 6 labels: 1
Mismatch.. (0, 2)
prob: 0.296937  class: 12 labels: 1
     prob: 0.16502  class: 2 labels: 38
     prob: 0.141899  class: 42 labels: 38
     prob: 0.0723655  class: 6 labels: 38
     prob: 0.0681367  class: 25 labels: 38
Mismatch.. (0, 3)
prob: 0.999551  class: 12 labels: 2
     prob: 0.000268922  class: 13 labels: 33
     prob: 0.000135727  class: 40 labels: 33
     prob: 1.21371e-05  class: 14 labels: 33
     prob: 1.07661e-05  class: 25 labels: 33
prob: 0.978146  class: 0 labels: 0
     prob: 0.0109876  class: 1 labels: 11
     prob: 0.00767156  class: 4 labels: 11
     prob: 0.00146429  class: 8 labels: 11
     prob: 0.000890397  class: 18 labels: 11
```

Addendum: notes to myself. Not for grading

4) skimage vs. opencv. skimage not work b/c of scaling images to 0-1 vs. 0-255. Can expand range to get this to work.
    skimage code test:

# X_train; add rotated, translated, brightness, contrast.

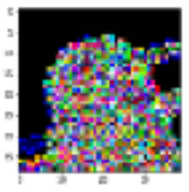
```
#what is layout. Get all images of class0 first
from skimage.transform import rotate

def rotImage(image,label):
    img=[]
    labels=[]
    rot_img = skimage.transform.rotate(image, 15, resize=True,
                         center=None, order=1, mode='edge', cval=0, clip=False,
preserve_range=True)
    img.append(rot_img)
    labels.append(label)
    rot_img1 = skimage.transform.rotate(image, -15, resize=True,
                         center=None, order=1, mode='edge', cval=0, clip=False,
preserve_range=True)
    img.append(rot_img1)
    labels.append(label)
    return img,labels

#rot_images, labels = rotImage(X_train[0],0)
#plt.imshow(rot)

rot = []
labels=[]
num = 0
for i in range(0,43):
    for idx, img in enumerate (X_train):
        if y_train[idx]==i:
            #print (idx)
            r,l = rotImage(img,i)
            rot.extend(r)
            labels.extend(l)
            num += 1
    print (len(rot), len(labels),num)
#print (len(rot), len(labels),num)
print(labels[10000])
plt.imshow(rot[10000])
```

This is the distorted 20km/h street sign with the skimage transform parameters. If we use default we don’t get this distortion. Was an attempt at testing effect of parameters.



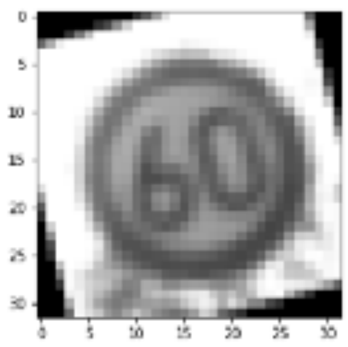
This shows a nondistorted skimage version. note the range of output between 0-1. Note: our normalziation contains 0 which is bad.

```
#0-1 displays same as 0-255. Range is messed up using skimage. Fix by expanding range and
compare to opencv
rot_np = np.array(rot)
print(np.amin(rot_np[10000]), np.amax(rot_np[10000]))

print(rot_np.shape)
rot_gray=rgb_gray(rot_np)
plt.imshow(rot_gray[10000],cmap='gray')
print(np.amin(rot_gray[10000]), np.amax(rot_gray[10000]))
print(np.amin(X_train[0]), np.amax(X_train[0]))

#rot_histo = apply_hist(rot_gray)
#plt.imshow(rot_histo[10000])
```

```
0.0 1.0
(69598, 32, 32, 3)
0.0 1.0
19 113
```



5) Experiment on opecv rotation. In color, the rotation produces black edge artifacts as the rotation angle increases. If you convert to grayscale this artifact is still there but is less visible.

Displays the progression showing the rotation, crop then resize. The rotation shifts the image. To remove the black edge artifacts you have to crop which increases the size; then expand back to 32x32 after the crop and reduce scale by .9 to negate the magnification effect. This was done by eyeball test. Not measured.

6) The output of TF it not deterministic. Follows a distribution. Run 10x and plot distribution. if this is a normal distribution we can use mean to describe the accuracy.