Traffic Sign Recognition Classifier

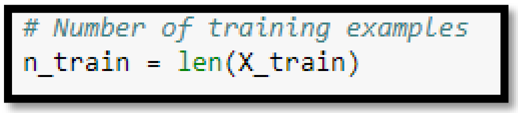
The goals / steps of this project are the following:

- Load the data set
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images

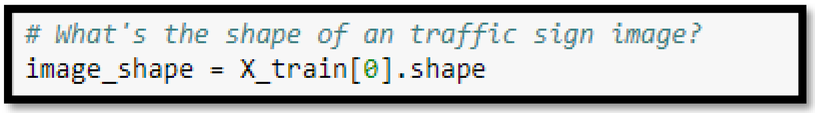
Step 1:

After loading the dataset, the first step was to calculate the summary statistics of the traffic signs data set. I used python's len() function to the calculate the number of training, validation, and test data. To calculate number of unique classes, I used python's set() function, which returns unique elements (duplicate elements are not allowed). I also calculated the image shape.

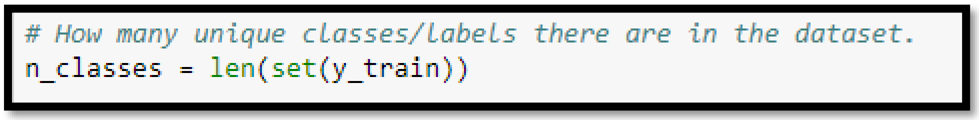
Number of training examples	34799
Number of validation examples	4410
Number of test examples	12630
Image data shape	(32, 32, 3)
Number of classes	43



```
# Number of training examples
n_train = len(X_train)
```



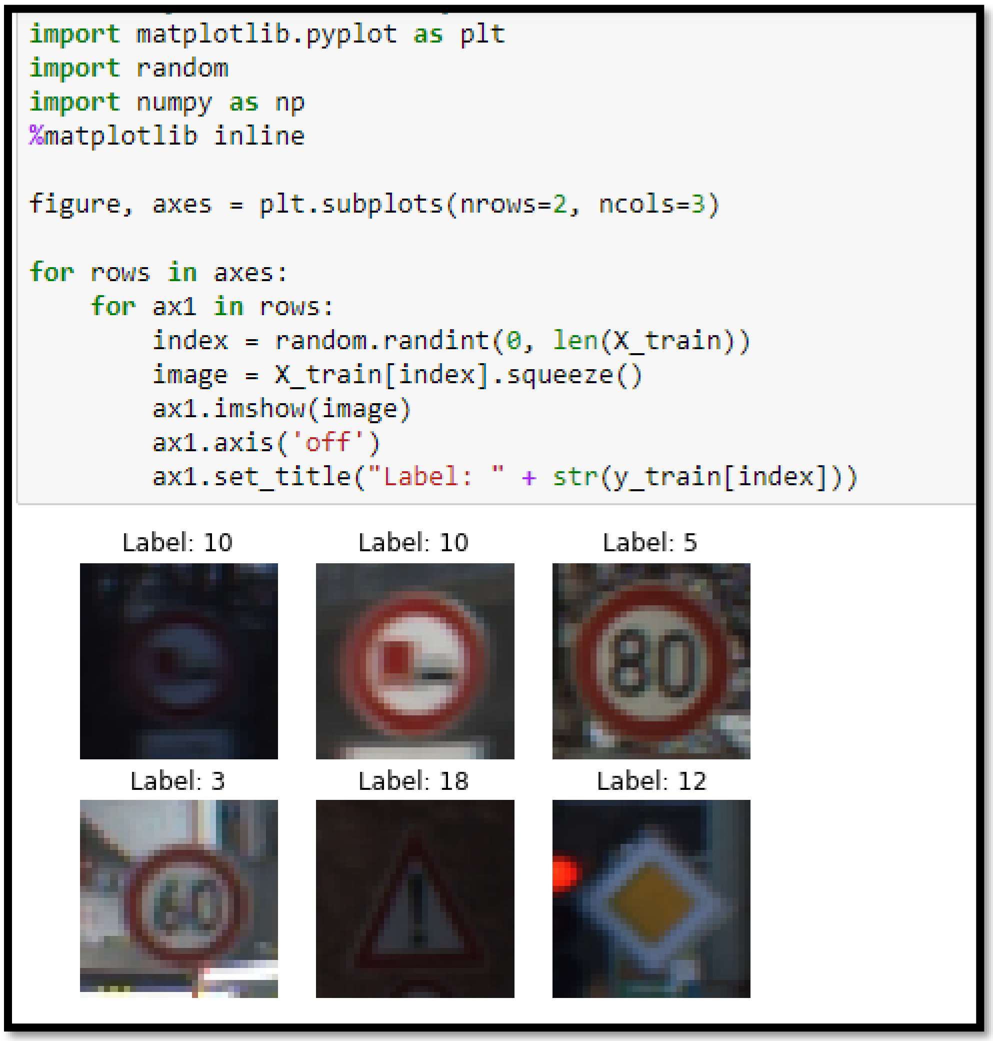
```
# What's the shape of an traffic sign image?
image_shape = X_train[0].shape
```



```
# How many unique classes/labels there are in the dataset.
n_classes = len(set(y_train))
```

Step 2:

For visualization, I simply used matplotlib to plot random 6 training images and printing their labels (nothing fancy).

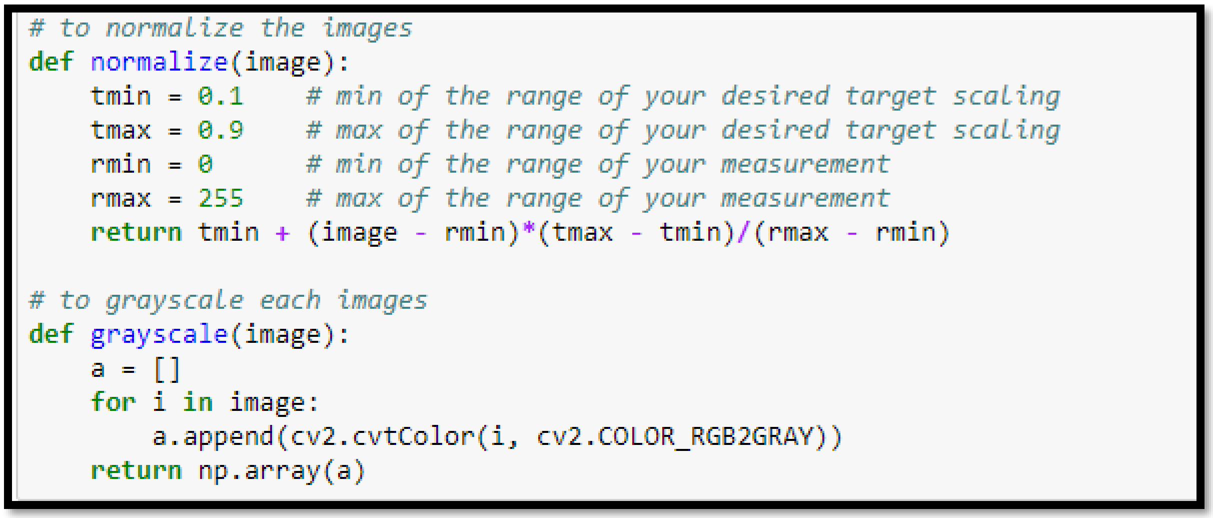


```
import matplotlib.pyplot as plt
import random
import numpy as np
%matplotlib inline

figure, axes = plt.subplots(nrows=2, ncols=3)

for rows in axes:
    for ax1 in rows:
        index = random.randint(0, len(X_train))
        image = X_train[index].squeeze()
        ax1.imshow(image)
        ax1.axis('off')
        ax1.set_title("Label: " + str(y_train[index]))
```

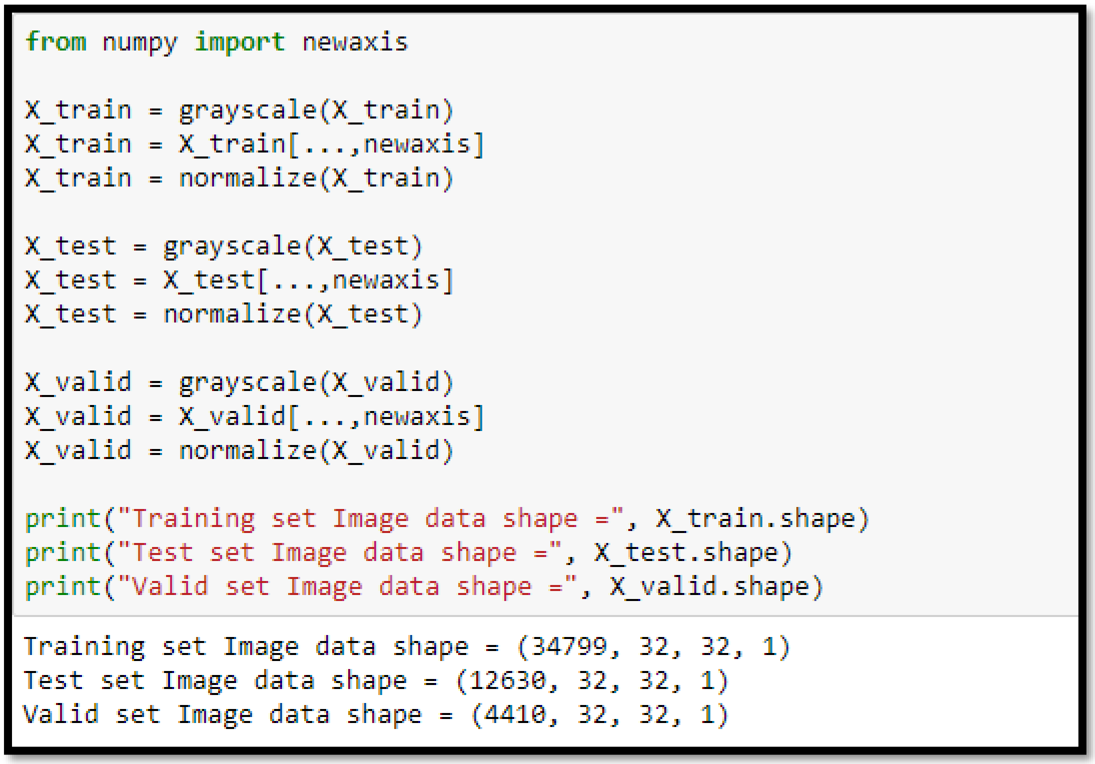
Step 3:

Preprocessing of the images, I tried – just normalizing (converting images to float and dividing by 255), and grayscaling images and normalizing (normalizing images within a particular range).

In the end, I chose the latter. I converted the RGB images to GRAYSCALE images. Used numpy's newaxis function to increase the dimension of the image after the conversion. (After converting the images to grayscale, the shape of the image came out to be (32, 32), so I used newaxis to convert to (32, 32, 1)). Then normalized the grayscaled images to a particular range (0.1 – 0.9).



```python
# to normalize the images
def normalize(image):
    tmin = 0.1    # min of the range of your desired target scaling
    tmax = 0.9    # max of the range of your desired target scaling
    rmin = 0      # min of the range of your measurement
    rmax = 255    # max of the range of your measurement
    return tmin + (image - rmin)*(tmax - tmin)/(rmax - rmin)

# to grayscale each images
def grayscale(image):
    a = []
    for i in image:
        a.append(cv2.cvtColor(i, cv2.COLOR_RGB2GRAY))
    return np.array(a)
```

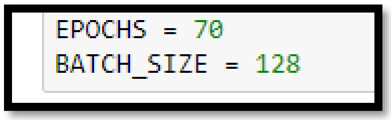


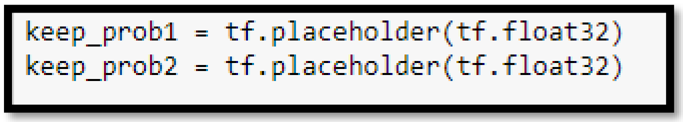
```python
from numpy import newaxis

X_train = grayscale(X_train)
X_train = X_train[...,newaxis]
X_train = normalize(X_train)

X_test = grayscale(X_test)
X_test = X_test[...,newaxis]
X_test = normalize(X_test)

X_valid = grayscale(X_valid)
X_valid = X_valid[...,newaxis]
X_valid = normalize(X_valid)

print("Training set Image data shape =", X_train.shape)
print("Test set Image data shape =", X_test.shape)
print("Valid set Image data shape =", X_valid.shape)
```

```
Training set Image data shape = (34799, 32, 32, 1)
Test set Image data shape = (12630, 32, 32, 1)
Valid set Image data shape = (4410, 32, 32, 1)
```

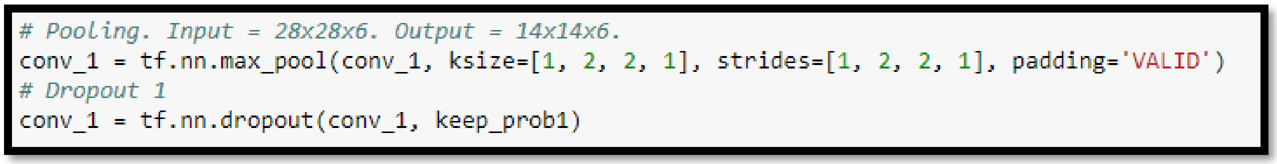
Step 4:

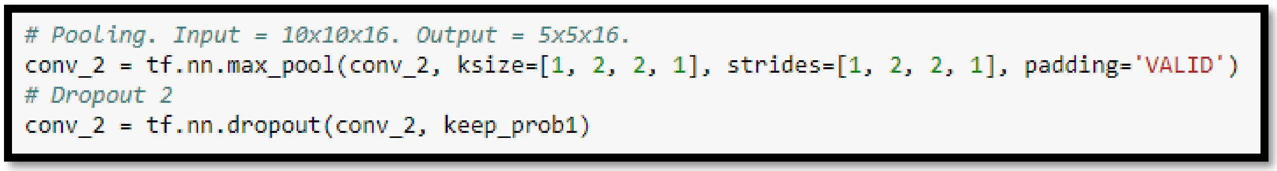
I used the LeNet architecture which was used for MNIST (LeNet Lab in TensorFlow). I added **dropout** after each max_pooling layers and after 1st fully_connected layer. Created placeholders for keep_prob. Set the number of EPOCHS to 70 and BATCH_SIZE to 128.

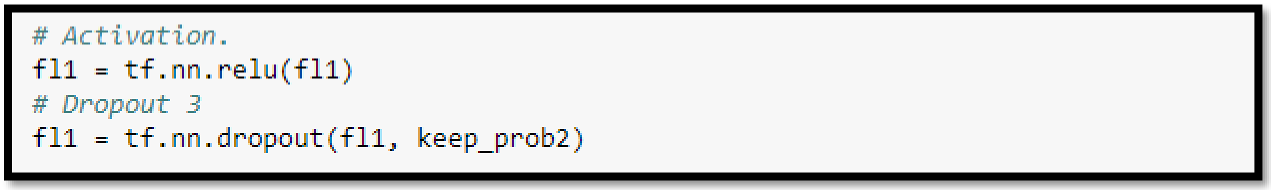

```
EPOCHS = 70
BATCH_SIZE = 128
```


```
keep_prob1 = tf.placeholder(tf.float32)
keep_prob2 = tf.placeholder(tf.float32)
```

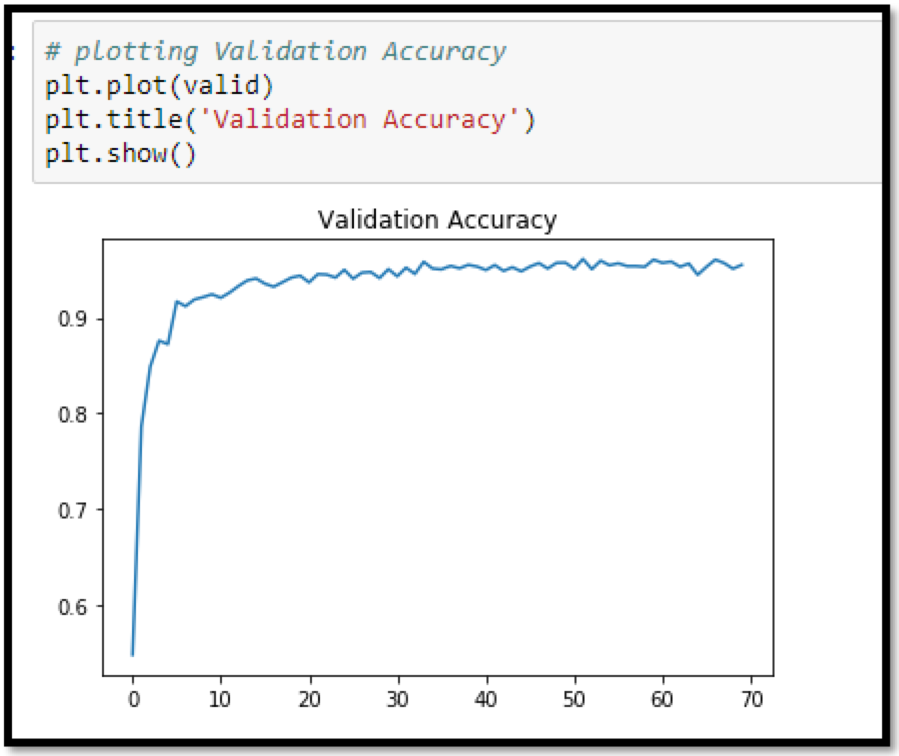
For the validation accuracy, I kept the dropout rate to 1, and for training, 0.8 and 0.7. I had to experiment on the dropout rates as to keep the validation accuracy about 0.93.


```
# Pooling. Input = 28x28x6. Output = 14x14x6.
conv_1 = tf.nn.max_pool(conv_1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
# Dropout 1
conv_1 = tf.nn.dropout(conv_1, keep_prob1)
```


```
# Pooling. Input = 10x10x16. Output = 5x5x16.
conv_2 = tf.nn.max_pool(conv_2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
# Dropout 2
conv_2 = tf.nn.dropout(conv_2, keep_prob1)
```

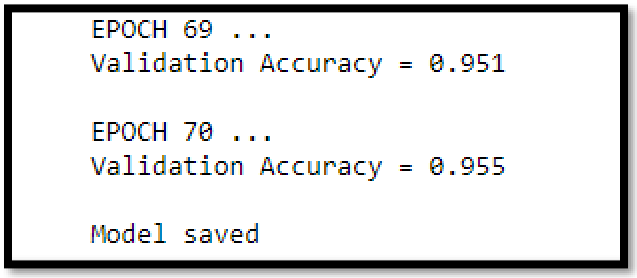

```
# Activation.
fl1 = tf.nn.relu(fl1)
# Dropout 3
fl1 = tf.nn.dropout(fl1, keep_prob2)
```

I created a list and appended all the validation accuracy in order to create a plot for Validation Accuracy.

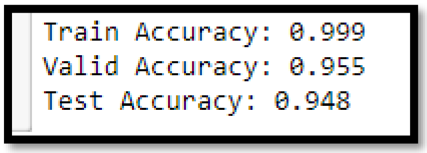


```
# plotting Validation Accuracy
plt.plot(valid)
plt.title('Validation Accuracy')
plt.show()
```

By adjusting the parameters, I was able to obtain a validation accuracy of 0.955 after 70 epochs.



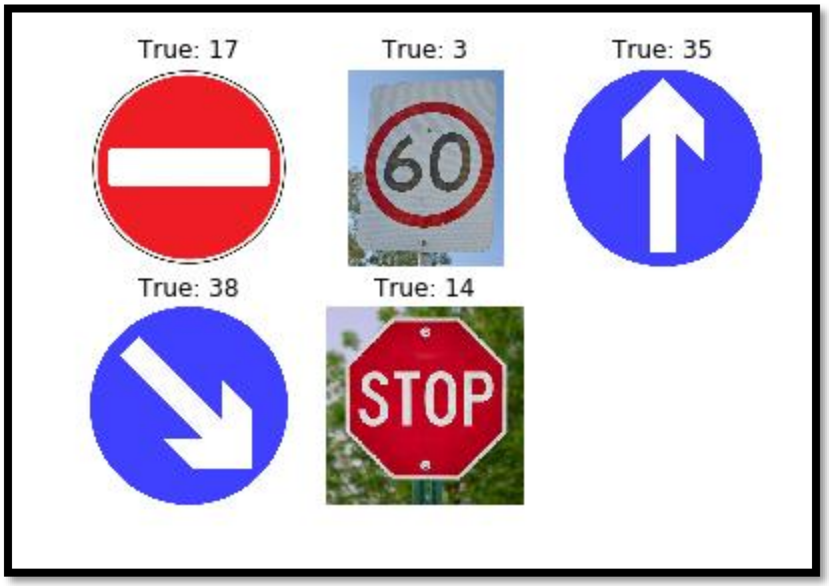
```
EPOCH 69 ...
Validation Accuracy = 0.951

EPOCH 70 ...
Validation Accuracy = 0.955

Model saved
```

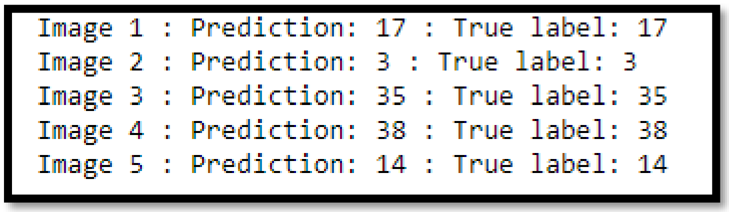


```
Train Accuracy: 0.999
Valid Accuracy: 0.955
Test Accuracy: 0.948
```

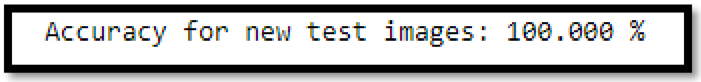
Step 5:

For testing new images, I collected 5 random test images, resized to (32, 32) for the model, preprocessed, and predicted the labels. I was able to get all the correct labels.





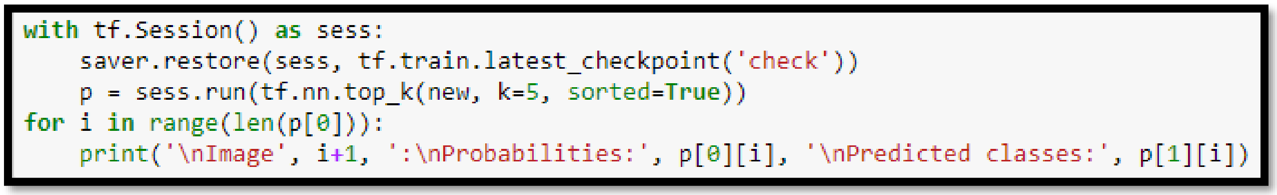
```
Image 1 : Prediction: 17 : True label: 17
Image 2 : Prediction: 3 : True label: 3
Image 3 : Prediction: 35 : True label: 35
Image 4 : Prediction: 38 : True label: 38
Image 5 : Prediction: 14 : True label: 14
```



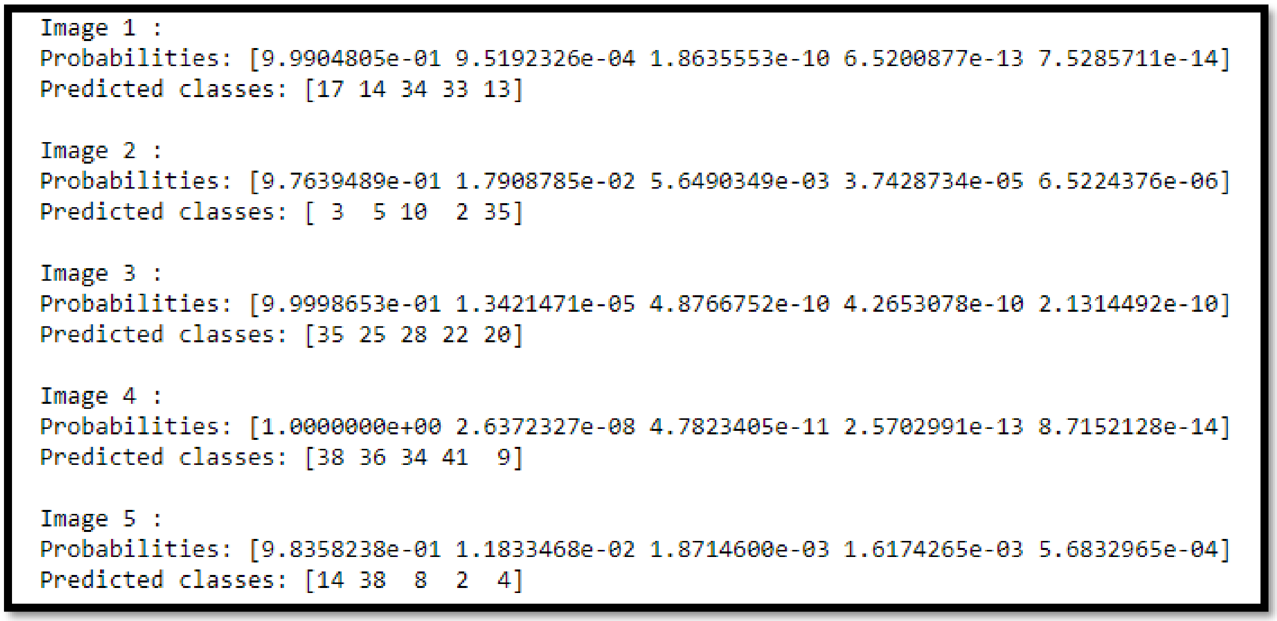
```
Accuracy for new test images: 100.000 %
```

Step 6:

To analyze the softmax probabilities for each image, I used **tf.nn.top_k** setting **k=5** to get top 5 probabilities.



```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('check'))
    p = sess.run(tf.nn.top_k(new, k=5, sorted=True))
for i in range(len(p[0])):
    print('\nImage', i+1, ':\nProbabilities:', p[0][i], '\nPredicted classes:', p[1][i])
```



```
Image 1 :
Probabilities: [9.9904805e-01 9.5192326e-04 1.8635553e-10 6.5200877e-13 7.5285711e-14]
Predicted classes: [17 14 34 33 13]

Image 2 :
Probabilities: [9.7639489e-01 1.7908785e-02 5.6490349e-03 3.7428734e-05 6.5224376e-06]
Predicted classes: [ 3  5 10  2 35]

Image 3 :
Probabilities: [9.9998653e-01 1.3421471e-05 4.8766752e-10 4.2653078e-10 2.1314492e-10]
Predicted classes: [35 25 28 22 20]

Image 4 :
Probabilities: [1.0000000e+00 2.6372327e-08 4.7823405e-11 2.5702991e-13 8.7152128e-14]
Predicted classes: [38 36 34 41  9]

Image 5 :
Probabilities: [9.8358238e-01 1.1833468e-02 1.8714600e-03 1.6174265e-03 5.6832965e-04]
Predicted classes: [14 38  8  2  4]
```