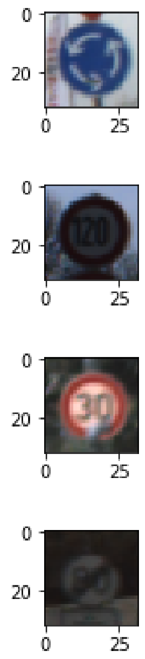
CarND-Traffic-Sign-Classifier- Writeup

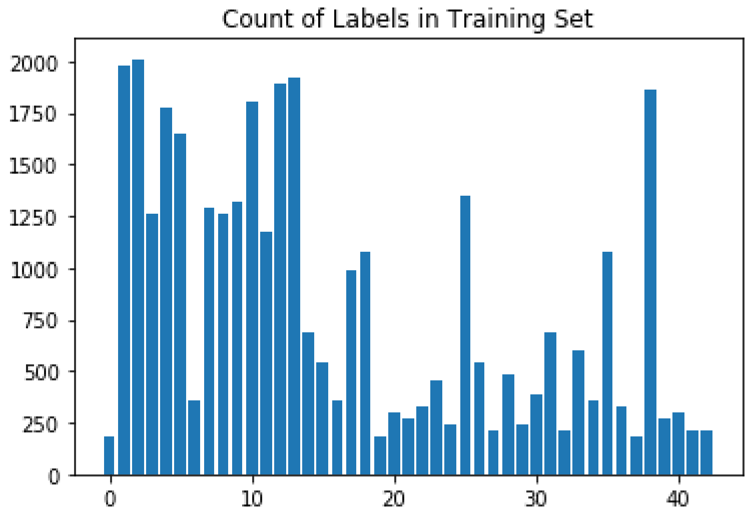
Step 1 a: Load images + count number of images + labels in each set

```
Length of training Set:  34799  Number of labels in Training Set:  34799
Length of Validation Set:  4410  Number of labels in Validation Set:  4410
Length of Test Set:  12630  Number of labels in Test Set:  12630
No of Unique labels:  43
```

Step 1b: Make sure the images are read in correctly :by displaying a few images and their labels in random from the training, validation, and test data sets + show the label distribution in the training set

```
Training Data Set
Label is:  40
Label is:  8
Label is:  1
Label is:  6
```





Step 2: Pre-process the input data images

Approaches tried: generate more data by converting color for the training set to HLS, HSV, rotating images by 90, 180 and 270 degrees, in addition to RGB and increase the number of images for training from 34799 to 34799* 6 images that are all normalized between -1 and 1 (Dropped this approach)

Grayscale conversion of input images and normalization between -1 and 1: (Preferred Approach as per notebook)

```
RGB Input Data Set Shape:  (34799, 32, 32, 3)
Normalized Data Set Shape:  (34799, 32, 32, 1)
```

```
RGB Input Data Set Shape:  (12630, 32, 32, 3)
Normalized Data Set Shape:  (12630, 32, 32, 1)
```

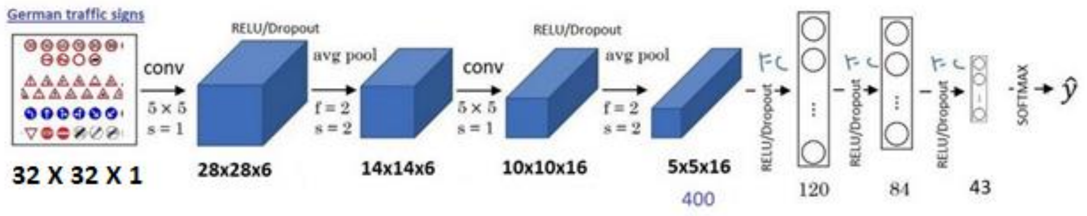
```
RGB Input Data Set Shape:  (4410, 32, 32, 3)
Normalized Data Set Shape:  (4410, 32, 32, 1)
```

Shuffle the input images before passing it to the next layer

Step3: Set up Model

Specs: Same as Lenet-5 with additional dropouts before each of the fully connected layers and 43 outputs

- Input image is 32X32X1 (also tried 32X32X3)
- Convolution layer 1. The output shape should be 28x28x6.
- Activation 1. Your choice of activation function.
- Pooling layer 1. The output shape should be 14x14x6.
- Convolution layer 2. The output shape should be 10x10x16.
- Activation 2. Your choice of activation function.
- Pooling layer 2. The output shape should be 5x5x16.
- Flatten layer.
- Dropout Layer: Added extra layer as suggested in the project notes
- Fully connected layer 1. This should have 120 outputs.
- Activation 3. Your choice of activation function.
- Dropout Layer: Added extra layer as suggested in the project notes
- Fully connected layer 2. This should have 84 outputs.
- Activation 4. Your choice of activation function.
- Dropout Layer: Added extra layer as suggested in the project notes
- Fully connected layer 3. This should have 43 outputs.

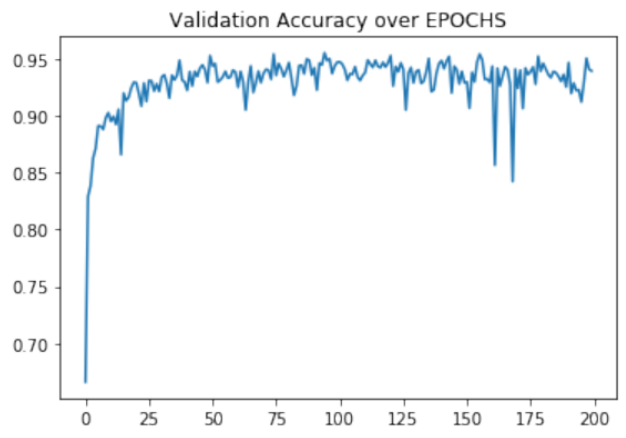


Architecture Diagram as provided in feedback with change only to input layer 32 32 1

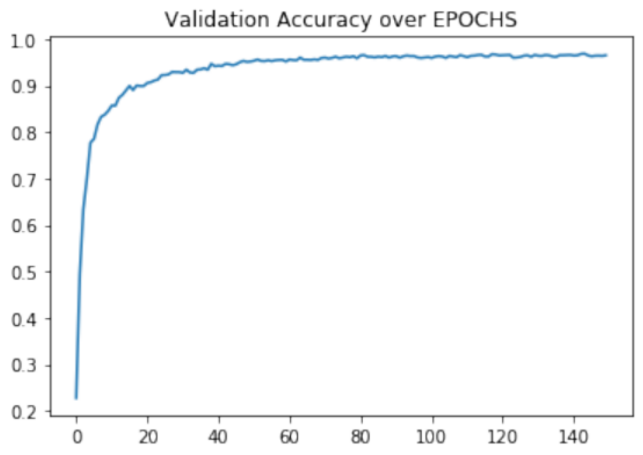
Step 4: Run Training and tune Hyper Parameters to improve validation accuracy

- Started with RGB:
    - Learning rate 0.008, Epochs 200, Keep Probability 0.5, Batch size 128
- Could not get training accuracy or validation accuracy > 75% with normalized color images and above parameters

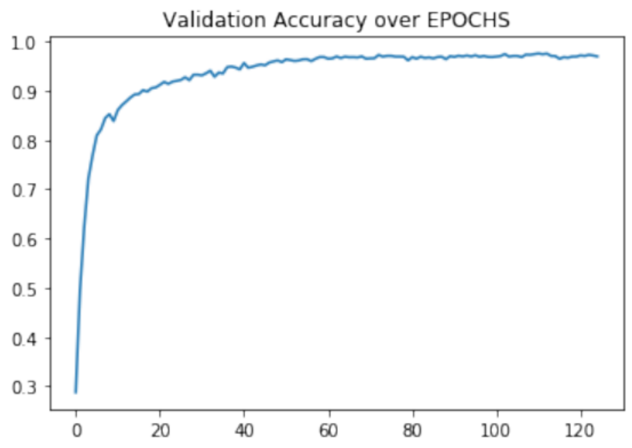
- Dropped learning rate to 0.005: Still ~80% training and validation accuracy
- **Move to Grayscale only approach:**
- Grayscale Attempts:
  - Learning rate 0.005, Epochs 200, Keep Probability 0.5, Batch size 128
  - Accuracy for validation set ~90% but does not settle



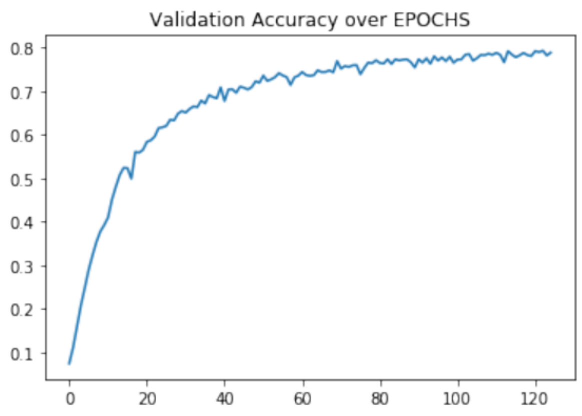
- Learning rate 0.0006, Epochs 150, Keep Probability 0.5, Batch size 128
- Accuracy for validation set ~95%



  -
- Learning rate 0.0005 Epochs 125, Keep Probability 0.5, Batch size 128
- Accuracy for validation set >95%



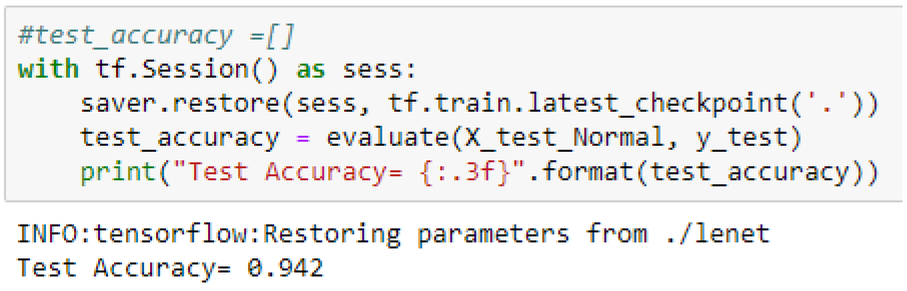
- **Finalized hyper parameters**
  - Learning rate 0.0005
  - Epochs 125
  - Keep Probability 0.5
  - Batch size 128
- Use the finalized hyper parameters on RGB normalized image instead of grayscale and see result:
  - Validation accuracy ~80% over the same



  - **Stay with Grayscale: re-run training with grayscale conversion instead**

Step 5: Run the saved model on test images:

- Test accuracy was 94.2% which is pretty close to the validation accuracy of ~95%



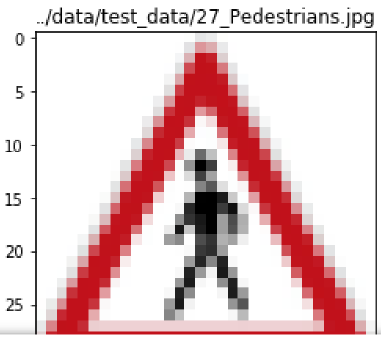
```
#test_accuracy =[]
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    test_accuracy = evaluate(X_test_Normal, y_test)
    print("Test Accuracy= {:.3f}".format(test_accuracy))
```

```
INFO:tensorflow:Restoring parameters from ./lenet
Test Accuracy= 0.942
```

Step 6: Download a new images from the web and predict labels:

- Downloaded images from: https://routetogermany.com/drivingingermany/road-signs
- Downloaded images from: https://www.rhinocarhire.com/Drive-Smart-Blog/Drive-Smart-Germany/Germany-Road-Signs.aspx
- Image selection criteria:
    - Image must be clear, avoided any image with pixilation issues before conversion
    - Images should have only one sign, none with multiple signs
    - The sign should be centered and should be viewed straight on
    - Images should be in color
    - Did not use images with text like speed limit signs that get hazy with re-size
    - Images should include only the sign and nothing in the background
    - Images must not have shadows and should be uniform brightness
    - Images must not be distortion free
    -
- Re-sized the images to match the input images that are expected by the normalizer function 32X32X3, save and display the images

```
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
```



- Predict labels for the images and calculate accuracy:
    - **Predicted Labels**

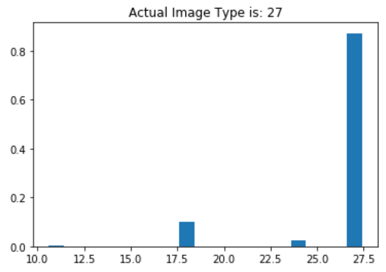
```
INFO:tensorflow:Restoring parameters from ./lenet
[27 13 25 17 14 11]
```

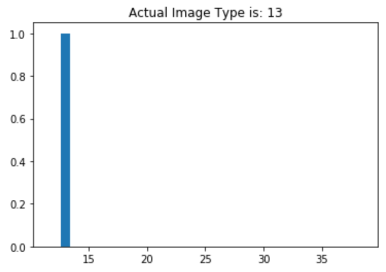
    - **Accuracy**

```
INFO:tensorflow:Restoring parameters from ./lenet
Model Accuracy for downloaded images = 1.000
```

Step 7: Display SoftMax for downloaded images to see how certain the model is for the downloaded images for the predicted labels:

```
In [82]: for i  in range (len(top_5[0])):
             plt.figure()
             plt.bar(top_5[1][i],top_5[0][i])
             plt.title('Actual Image Type is: '+str(Y_down[i]))
```





•