Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

In this notebook, a template is provided for you to implement your functionality in stages, which is required to successfully complete this project. If additional code is required that cannot be included in the notebook, be sure that the Python code is successfully imported and included in your submission if necessary.

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the iPython Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there is a writeup to complete. The writeup should be completed in a separate file, which can be either a markdown file or a pdf document. There is a write up template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) that can be used to guide the writing process. Completing the code template and writeup template will cover all of the rubric points (https://review.udacity.com/#!/rubrics/481/view) for this project.

The rubric (https://review.udacity.com/#!/rubrics/481/view) contains "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. The stand out suggestions are optional. If you decide to pursue the "stand out suggestions", you can include the code in this Ipython notebook and also discuss the results in the writeup file.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Step 0: Load The Data

In [28]:

```
from __future__ import print_function
import pickle
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import Conv_NN as cnn
from pathlib import Path
import logging
import sys
import cv2

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

training_file = "./traffic-signs-data/train.p"
validation_file = "./traffic-signs-data/valid.p"
testing_file = "./traffic-signs-data/test.p"

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

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

Step 1: Dataset Summary & Exploration

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 1D 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 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**

Complete the basic data summary below. Use python, numpy and/or pandas methods to calculate the data summary rather than hard coding the results. For example, the [pandas shape method (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html) might be useful for calculating some of the summary results.

Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

In [29]:

```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results

# TODO: Number of training examples
n_train = train['features'].shape[0]

# TODO: Number of validation examples
n_validation = valid['features'].shape[0]

# TODO: Number of testing examples.
n_test = test['features'].shape[0]

# TODO: What's the shape of an traffic sign image?
image_shape = [train['features'].shape[1], train['features'].shape[2], train['features'
].shape[3]]

# TODO: How many unique classes/labels there are in the dataset.
n_classes = np.max(train['labels'])+1

print("Number of training examples =", n_train)
print("Number of validation examples =", n_validation)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = [32, 32, 3]
Number of classes = 43
```

Include an exploratory visualization of the dataset

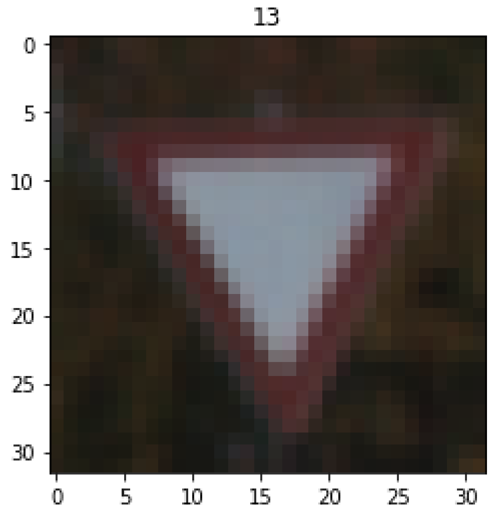
Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

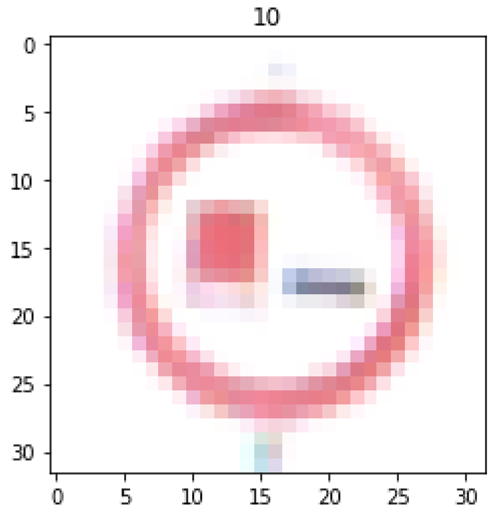
The Matplotlib (http://matplotlib.org/) examples (http://matplotlib.org/examples/index.html) and gallery (http://matplotlib.org/gallery.html) pages are a great resource for doing visualizations in Python.

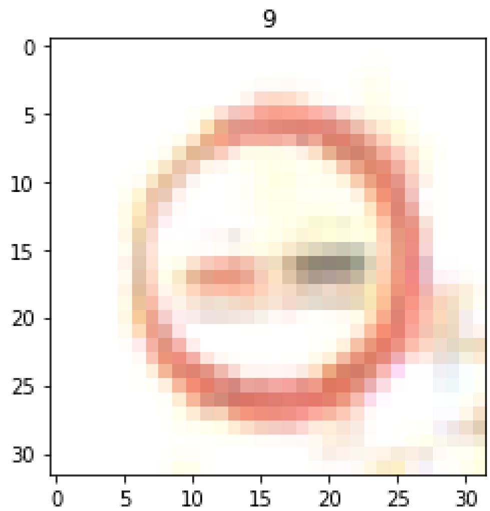
NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples of some classes than others?

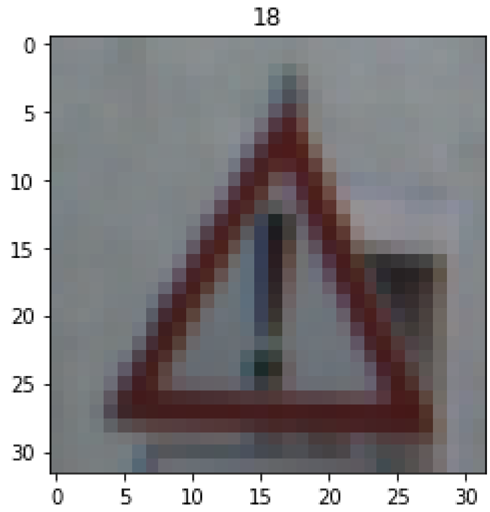
In [30]:

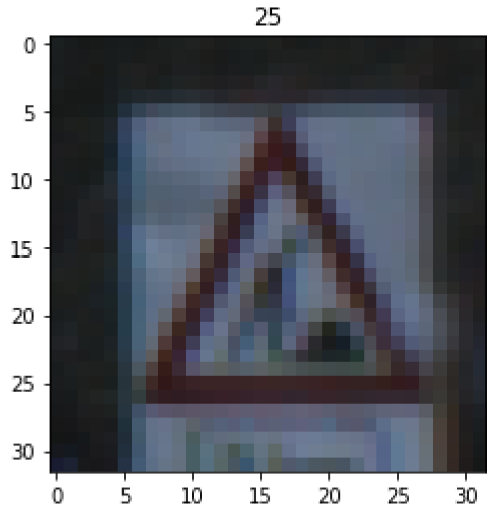
```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline

# Plot Randon data
idx = np.random.permutation(train['features'].shape[0])[:5]
for i in idx:
    im = np.uint8((train['features'][i,:,:,:]))
    plt.title(train['labels'][i])
    plt.imshow(im)
    plt.show()

# ----------------------------------------------------
# plot the number of examples of training
unique, counts = np.unique(train['labels'], return_counts=True)
plt.bar(unique, counts, 1/1.5, color="green")
plt.title("Number of samples per class Training")
plt.xlabel("Class")
plt.ylabel("Number of samples")
plt.show()
# ----------------------------------------------------
# plot the number of examples of Validation
unique, counts = np.unique(valid['labels'], return_counts=True)
plt.bar(unique, counts, 1/1.5, color="red")
plt.title("Number of samples per class Validation")
plt.xlabel("Class")
plt.ylabel("Number of samples")
plt.show()
# ----------------------------------------------------
# plot the number of examples of Test
unique, counts = np.unique(test['labels'], return_counts=True)
plt.bar(unique, counts, 1/1.5, color="blue")
plt.title("Number of samples per class Test")
plt.xlabel("Class")
plt.ylabel("Number of samples")
plt.show()

#----------------------------------------------------
```

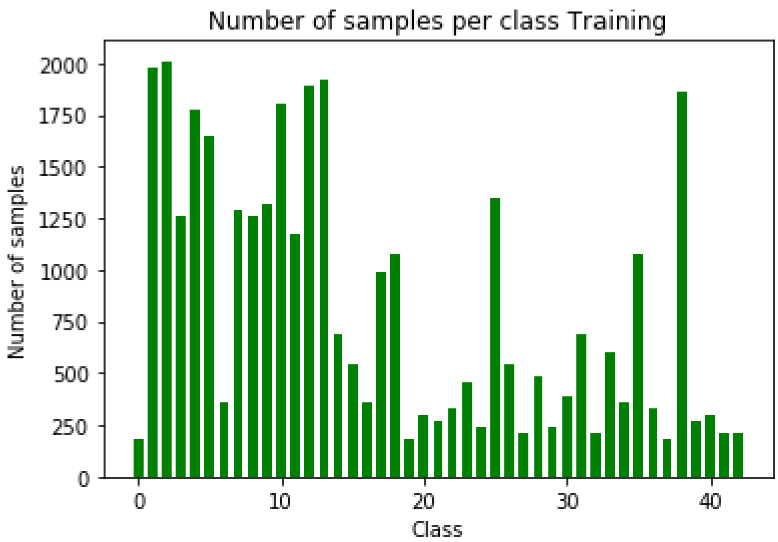

13

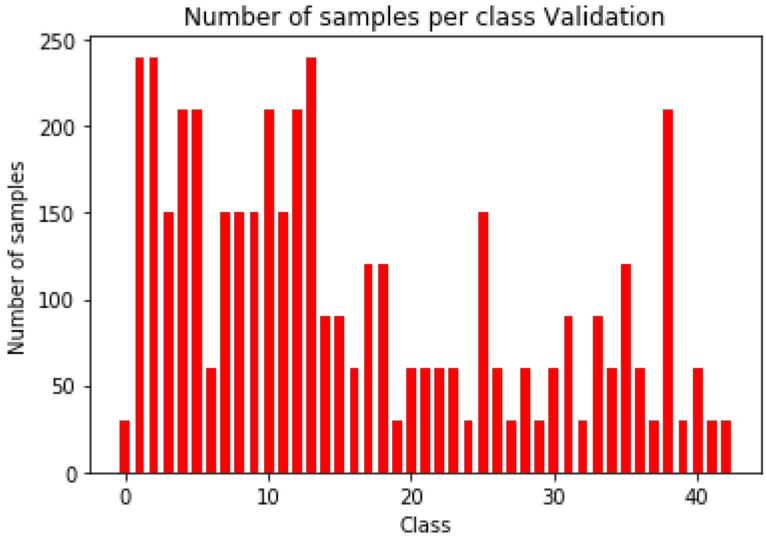

10

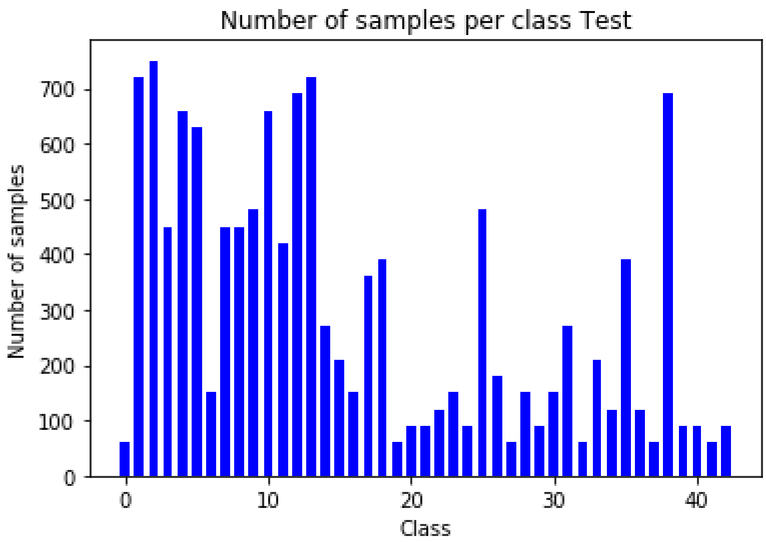

9


18


25


Number of samples per class Training
Number of samples
Class


Number of samples per class Validation
Number of samples
Class
0
50
100
150
200
250
0
10
20
30
40


Number of samples per class Test
Number of samples
Class
0
100
200
300
400
500
600
700
0
10
20
30
40

Step 2: Design and Test a Model Architecture

Design and implement a deep learning model that learns to recognize traffic signs. Train and test your model on the German Traffic Sign Dataset (http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset).

The LeNet-5 implementation shown in the classroom (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) at the end of the CNN lesson is a solid starting point. You'll have to change the number of classes and possibly the preprocessing, but aside from that it's plug and play!

With the LeNet-5 solution from the lecture, you should expect a validation set accuracy of about 0.89. To meet specifications, the validation set accuracy will need to be at least 0.93. It is possible to get an even higher accuracy, but 0.93 is the minimum for a successful project submission.

There are various aspects to consider when thinking about this problem:

- Neural network architecture (is the network over or underfitting?)
- Play around preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others).
- Generate fake data.

Here is an example of a published baseline model on this problem (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf). It's not required to be familiar with the approach used in the paper but, it's good practice to try to read papers like these.

In [32]:

```
# ------------------------------------------------------------------------------------
### Pre-process the Data Set
# ------------------------------------------------------------------------------------

def normalize_data(dataset):
  dataset = dataset.astype(np.float32)
  # Grayscale,It is not the best way but the faster
  dataset = np.uint8(np.sum(dataset / 3, axis=3, keepdims=True))
  # Equalization of the image
  for i in range(dataset.shape[0]):
      dataset[i, :, :,0] =  cv2.equalizeHist(dataset[i, :, :,0])
  # Normalization
  dataset = (dataset/255)-0.5
  return dataset
# ------------------------------------------------------------------------------------
def create_hot_ones(labels, num_labels):
  labels = (np.arange(num_labels) == labels[:, None]).astype(np.float32)
  return labels
# ------------------------------------------------------------------------------------
## Normalize the images and generate the hot-ones
X_train, Y_train = normalize_data(train['features']), create_hot_ones(train['labels'],
n_classes)
X_valid, Y_valid = normalize_data(valid['features']), create_hot_ones(valid['labels'],
n_classes)
X_test, Y_test = normalize_data(test['features']), create_hot_ones(test['labels'], n_cl
asses)
```

In [33]:

```
# Set the number of channel to one
image_shape[2] = 1
```

In [34]:

```
# Save the data to create the augmentation
print("Save")
# ------------------------------------------------------------------------
# Train
# ------------------------------------------------------------------------
with open('X_train_Normalized.pickle', 'wb') as handle:
     pickle.dump(X_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('Y_train_Normalized.pickle', 'wb') as handle:
     pickle.dump(Y_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
# ------------------------------------------------------------------------
# Validation
# ------------------------------------------------------------------------
with open('X_valid_Normalized.pickle', 'wb') as handle:
     pickle.dump(X_valid, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('Y_valid_Normalized.pickle', 'wb') as handle:
     pickle.dump(Y_valid, handle, protocol=pickle.HIGHEST_PROTOCOL)
# ------------------------------------------------------------------------
# Test
# ------------------------------------------------------------------------
with open('X_test_Normalized.pickle', 'wb') as handle:
     pickle.dump(X_test, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('Y_test_Normalized.pickle', 'wb') as handle:
     pickle.dump(Y_test, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
Save
```

Create the augmentation data

This code is based on

In [ ]:

```
import numpy as np
import pickle
import matplotlib.pyplot as plt
from pathlib import Path
import time
import cv2
# ---------------------------------------------------------------------------
def augment_brightness_camera_images(image):

    image1 = np.array(image, dtype = np.float64)
    random_bright = .5+np.random.uniform()
    image1[:,:] = image1[:,:]*random_bright
    image1[:,:][image1[:,:]>255] = 255
    image1 = np.array(image1, dtype=np.uint8)
    return image1
# ---------------------------------------------------------------------------
def transform_image(image, ang_range, shear_range, trans_range):
    # Rotation
    ang_rot = np.random.uniform(ang_range) - ang_range / 2
    rows, cols, ch = image.shape
    Rot_M = cv2.getRotationMatrix2D((cols / 2, rows / 2), ang_rot, 1)
    # Translation
    tr_x = trans_range * np.random.uniform() - trans_range / 2
    tr_y = trans_range * np.random.uniform() - trans_range / 2
    Trans_M = np.float32([[1, 0, tr_x], [0, 1, tr_y]])
```

```
    # Shear
    pts1 = np.float32([[5, 5], [20, 5], [5, 20]])
    pt1 = 5 + shear_range * np.random.uniform() - shear_range / 2
    pt2 = 20 + shear_range * np.random.uniform() - shear_range / 2
    pts2 = np.float32([[pt1, 5], [pt2, pt1], [5, pt2]])
    shear_M = cv2.getAffineTransform(pts1, pts2)

    image = cv2.warpAffine(image, Rot_M, (cols, rows))
    image = cv2.warpAffine(image, Trans_M, (cols, rows))
    image = cv2.warpAffine(image, shear_M, (cols, rows))

    # Brightness augmentation
    image = augment_brightness_camera_images(image)

    return image

#-------------------------------------------------------------------------------------------
with open('X_train_Normalized.pickle', 'rb') as handle:
    X_train = pickle.load(handle)
with open('y_train_Normalized.pickle', 'rb') as handle:
    y_train = pickle.load(handle)

# convert hot ones to classes
classes = [np.where(r == 1)[0][0] for r in y_train]
# plot the number of examples of training
unique, counts = np.unique(classes, return_counts=True)
classes_array = np.array(classes)

First_time = True
for class_id in range(43):
    class_indexs = np.argwhere(classes_array == class_id)
    X_train_Augmented = (X_train[class_indexs, :, :, :])
    y_train_Augmented = (y_train[class_indexs])

    X_train_Augmented = np.squeeze(X_train_Augmented, axis=1)
    y_train_Augmented = np.squeeze(y_train_Augmented, axis=1)
    not_filled = True

    while not_filled is True:
        for id_image in class_indexs:

            if counts[class_id] < 4200:
               First_time = False

               im = np.uint8((X_train[id_image,:,:,:]+0.5)*255)
               im = np.squeeze(im, axis=0)

               if class_id == 8:
                   pp = 0

               img_aug = transform_image(im, 30, 5, 5)
               img_aug = np.expand_dims(img_aug, axis=2)


               X_train_Augmented = np.concatenate([X_train_Augmented, [(img_aug / 255.0
)-0.5]])
               y_train_Augmented = np.concatenate([y_train_Augmented, y_train[id_image
]])

               counts[class_id] += 1
```

```
            else:
                if First_time is True:

                    X = X_train[class_indexs]
                    X = np.squeeze(X, axis=1)
                    X_train_Augmented = X

                    Y = y_train[class_indexs]
                    Y = np.squeeze(Y, axis=1)
                    y_train_Augmented = Y

                not_filled = False
                print("Filled Class: ", class_id)
                print("Number of elements: ", counts[class_id])
                First_time = True

                with open('./Sub_Augmentation/X_train_Augmented_sub'+str(class_id)+'.pic
kle', 'wb') as handle:
                    pickle.dump(X_train_Augmented, handle, protocol=pickle.HIGHEST_PROTO
COL)
                with open('./Sub_Augmentation/Y_train_Augmented_sub'+str(class_id)+'.pic
kle', 'wb') as handle:
                    pickle.dump(y_train_Augmented, handle, protocol=pickle.HIGHEST_PROTO
COL)

                X_train_Augmented = np.array([X_train[id_image, :, :, :]])
                y_train_Augmented = np.array([y_train[id_image]])

                X_train_Augmented = np.squeeze(X_train_Augmented, axis=1)
                y_train_Augmented = np.squeeze(y_train_Augmented, axis=1)
                break
    #---------------------------------------------------------------------------------
-------
```

Fuse the classes into a single file

In [24]:

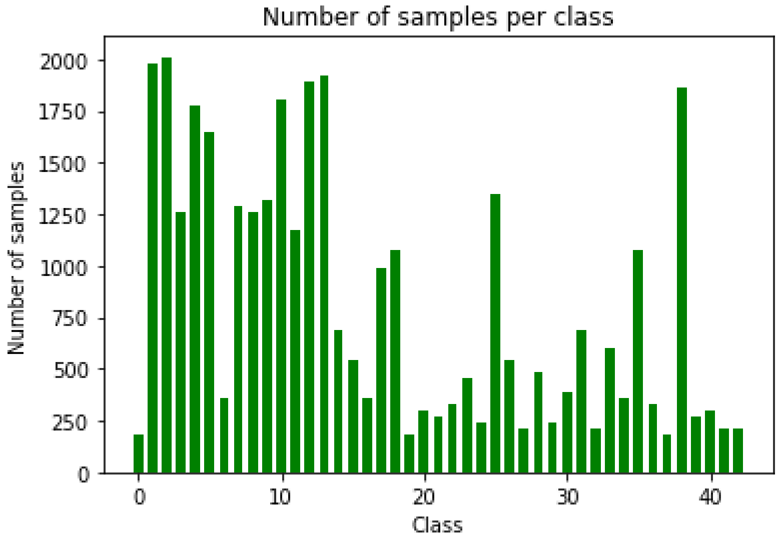
```
import numpy as np
import pickle
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
# ----------------------------------------------------------------------
i=0
print("Subset: ",i)
with open('Sub_Augmentation/X_train_Augmented_sub' + str(i) + '.pickle', 'rb') as handl
e:
    X_train_Augmentation = pickle.load(handle)
with open('Sub_Augmentation/Y_train_Augmented_sub' + str(i) + '.pickle', 'rb') as handl
e:
    Y_train_Augmentation = pickle.load(handle)

for i in range(1,43):
    print("Subset: ",i)
    with open('Sub_Augmentation/X_train_Augmented_sub' + str(i) + '.pickle', 'rb') as h
andle:
        X_train_ = pickle.load(handle)
    with open('Sub_Augmentation/Y_train_Augmented_sub' + str(i) + '.pickle', 'rb') as h
andle:
        y_train_ = pickle.load(handle)

    X_train_Augmentation = np.concatenate([X_train_Augmentation, X_train_])
    Y_train_Augmentation = np.concatenate([Y_train_Augmentation, y_train_])

with open('X_train_Augmented_Final.pickle', 'wb') as handle:
     pickle.dump(X_train_Augmentation, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('Y_train_Augmented_Final.pickle', 'wb') as handle:
    pickle.dump(Y_train_Augmentation, handle, protocol=pickle.HIGHEST_PROTOCOL)

# Shuffle the data
X_train_Augmentation, Y_train_Augmentation = shuffle(X_train_Augmentation, Y_train_Augm
entation)

with open('X_train_Augmented_Final_Shuffle.pickle', 'wb') as handle:
    pickle.dump(X_train_Augmentation, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('Y_train_Augmented_Final_Shuffle.pickle', 'wb') as handle:
    pickle.dump(Y_train_Augmentation, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

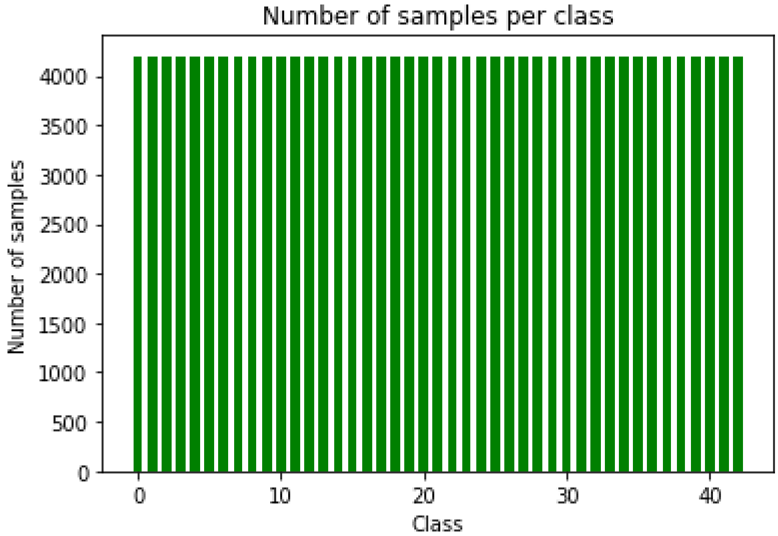
```
Subset:  0
Subset:  1
Subset:  2
Subset:  3
Subset:  4
Subset:  5
Subset:  6
Subset:  7
Subset:  8
Subset:  9
Subset:  10
Subset:  11
Subset:  12
Subset:  13
Subset:  14
Subset:  15
Subset:  16
Subset:  17
Subset:  18
Subset:  19
Subset:  20
Subset:  21
Subset:  22
Subset:  23
Subset:  24
Subset:  25
Subset:  26
Subset:  27
Subset:  28
Subset:  29
Subset:  30
Subset:  31
Subset:  32
Subset:  33
Subset:  34
Subset:  35
Subset:  36
Subset:  37
Subset:  38
Subset:  39
Subset:  40
Subset:  41
Subset:  42
```

Plot the classes histogram

In [35]:

```
#------------------------------------------------------------------------------------------
# Plot Histogram
#------------------------------------------------------------------------------------------
def plt_histogram(y_train):

    classes = [np.where(r == 1)[0][0] for r in y_train]
    # plot the number of examples of training
    unique, counts = np.unique(classes, return_counts=True)
    plt.bar(unique, counts, 1 / 1.5, color="green")
    plt.title("Number of samples per class")
    plt.xlabel("Class")
    plt.ylabel("Number of samples")
    plt.show()
    return counts
#------------------------------------------------------------------------------------------
# Original
plt_histogram(Y_train)
# After Augmentation
plt_histogram(Y_train_Augmentation)
```





Out[35]:

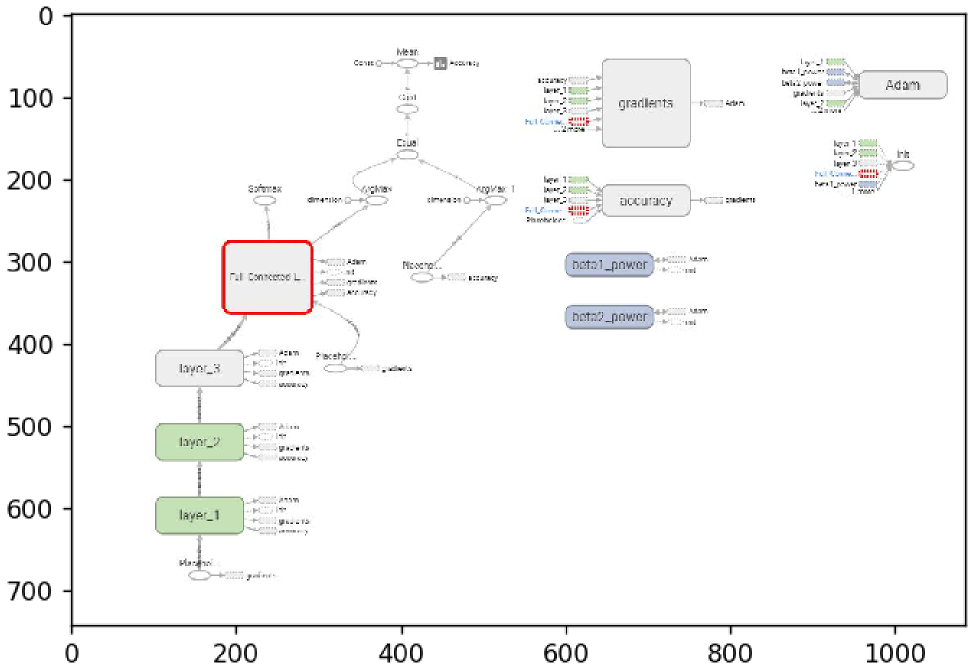
```
array([4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200,
       4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200,
       4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200,
       4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200, 4200], dtype=
int64)
```

Model Architecture

In [20]:

```
%matplotlib notebook
print("Global architecture")
im=plt.imread('./images/global_graph.JPG')
plt.figure()
plt.imshow(im)
plt.show()
```

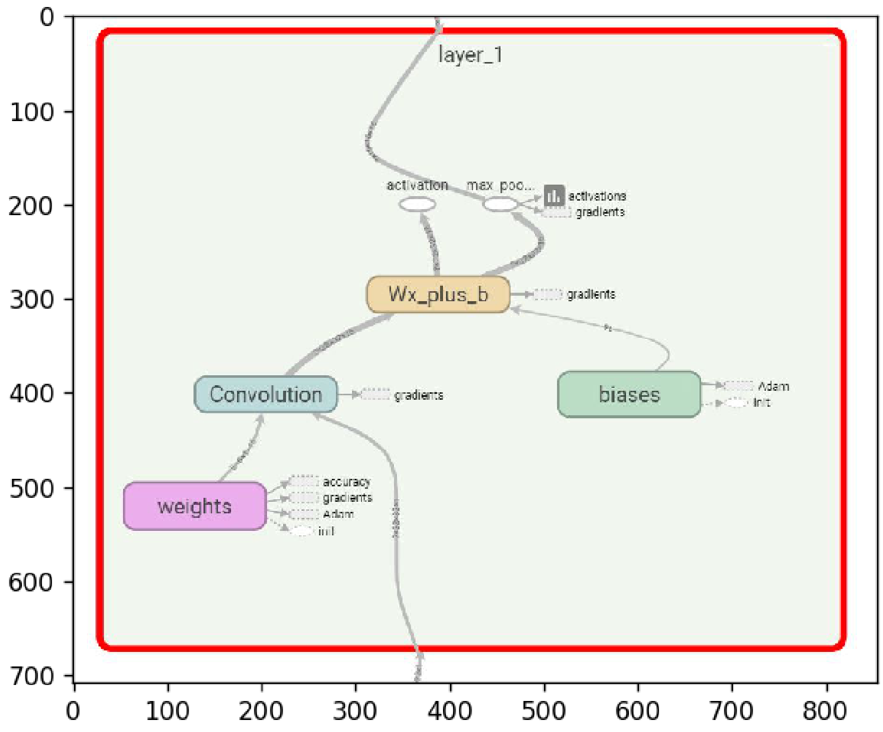
```
Global architecture
```



In [21]:

```
%matplotlib notebook
print("Layer architecture")
im=plt.imread('./images/convolution_layer.JPG')
plt.figure()
plt.imshow(im)
plt.show()
```

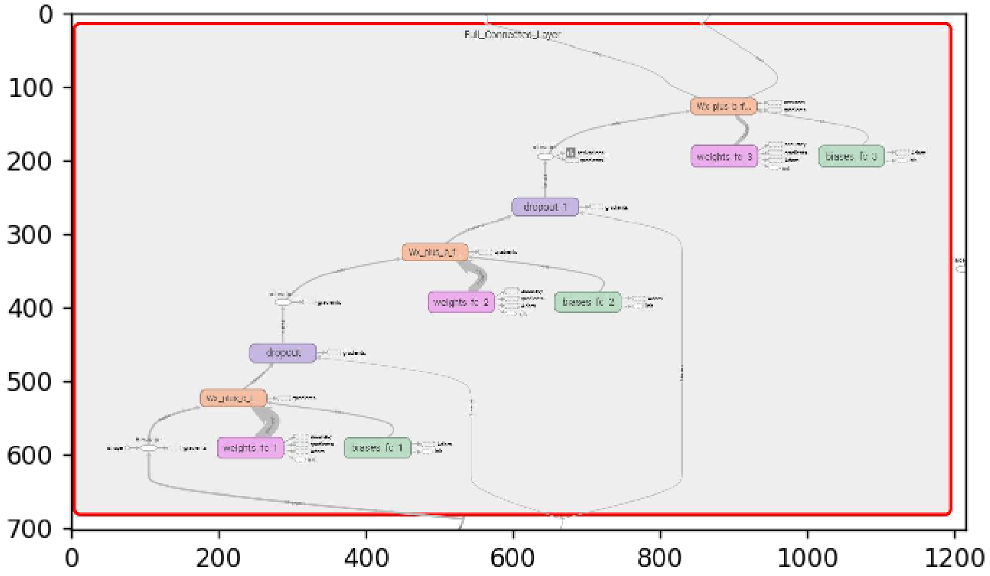
Layer architecture



In [22]:

```
%matplotlib notebook
print("Full Connected layer architecture")
im=plt.imread('./images/fc_layer.JPG')
plt.figure()
plt.imshow(im)
plt.show()
```

```
Full Connected layer architecture
```



The architecture used in this exercise is a Lenet5 with three convolution layer and a full connected layer.

CNN Layer	Filter Size	Num. Filters
1	5x5	16
2	5x5	32
3	3x3	64

Full Connect Layer size: 1024

In [7]:

```
#Functions to train the Network
import tensorflow as tf
#-------------------------------------------------------------------------------
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.01)
    return tf.Variable(initial, name="weight_")
#-------------------------------------------------------------------------------
def bias_variable(shape):
    initial = tf.constant(0.0, shape=shape)
```

```
    return tf.Variable(initial, name="bias_")
#------------------------------------------------------------------------------------------
def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)
#------------------------------------------------------------------------------------------
def cnn_layer(input_tensor, layer_name,filter_size,num_dimension,num_filters,dropout_
value, max_pooling, act=tf.nn.relu):
  """Reusable code for making a simple neural net layer.

  It does a matrix multiply, bias add, and then uses relu to nonlinearize.
  It also sets up name scoping so that the resultant graph is easy to read,
  and adds a number of summary ops.
  """
  # ------------------------------------------------------------------------------------
-
  # Adding a name scope ensures logical grouping of the layers in the graph.
  # ------------------------------------------------------------------------------------
-
  with tf.name_scope(layer_name):
    # ----------------------------------------------------------------------------------
---
    # This Variable will hold the state of the weights for the layer
    # ----------------------------------------------------------------------------------
---
    with tf.name_scope('weights'):
      weights = weight_variable([filter_size, filter_size, num_dimension, num_filters
])
      variable_summaries(weights)
    # ----------------------------------------------------------------------------------
---
    # Bias
    # ----------------------------------------------------------------------------------
---
    with tf.name_scope('biases'):
      biases = bias_variable([num_filters])
      variable_summaries(biases)
    # ----------------------------------------------------------------------------------
---
    # Convolution
    # ----------------------------------------------------------------------------------
---
    with tf.name_scope('Convolution'):
        convolution = tf.nn.conv2d(input_tensor, weights, strides=[1, 1, 1, 1], paddi
ng='SAME', use_cudnn_on_gpu=True, name="Convolution")
    # ----------------------------------------------------------------------------------
---
    # Bias Addition to the convolution
    # ----------------------------------------------------------------------------------
---
    with tf.name_scope('Wx_plus_b'):
      preactivate = tf.add(convolution, biases)
      tf.summary.histogram('pre_activations', preactivate)
```

```
    # -----------------------------------------------------------------------------------
---
    # Activation
    # -----------------------------------------------------------------------------------
--
    activations = act(preactivate, name='activation')
    if max_pooling is True:
        result = tf.nn.max_pool(preactivate, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1
], padding='SAME', name='max_pool_1')
    else:
        result = activations
    tf.summary.histogram('activations', result)
    return result
# -----------------------------------------------------------------------------------
def full_connect_layer(input_tensor, layer_name, size_input, size_FC, num_ouputs, dro
pout_value,is_flat, act=tf.nn.relu, act2=tf.nn.relu):

    # ---------------------------------------------------------------------
    # Create the Weight 1
    # ---------------------------------------------------------------------
    with tf.name_scope(layer_name):
        with tf.name_scope('weights_fc_1'):
          weights_fc_1 = weight_variable([size_input, size_FC])
          variable_summaries(weights_fc_1)
        # ---------------------------------------------------------------------
        # Create the Bias 1
        # ---------------------------------------------------------------------
        with tf.name_scope('biases_fc_1'):
          biases_fc_1 = bias_variable([size_FC])
          variable_summaries(biases_fc_1)
        # ---------------------------------------------------------------------
        # Create the Weight 2
        # ---------------------------------------------------------------------
        with tf.name_scope('weights_fc_2'):
          weights_fc_2 = weight_variable([size_FC, size_FC])
          variable_summaries(weights_fc_2)
        # ---------------------------------------------------------------------
        # Create the Bias 2
        # ---------------------------------------------------------------------
        with tf.name_scope('biases_fc_2'):
          biases_fc_2 = bias_variable([size_FC])
          variable_summaries(biases_fc_2)
        # ---------------------------------------------------------------------------
------------------
        # # ---------------------------------------------------------------------
        # # Create the Weight 3
        # # ---------------------------------------------------------------------
        with tf.name_scope('weights_fc_3'):
             weights_fc_3 = weight_variable([size_FC, num_ouputs])
             variable_summaries(weights_fc_3)
        # # # ---------------------------------------------------------------------
        # # # Create the Bias 3
        # # # ---------------------------------------------------------------------
        with tf.name_scope('biases_fc_3'):
             biases_fc_3 = bias_variable([num_ouputs])
             variable_summaries(biases_fc_3)
        # ---------------------------------------------------------------------------
------------------
        # Reshape the input
        # ---------------------------------------------------------------------------
------------------
```

```
        if is_flat is False:
           input_tensor_reshaped = tf.reshape(input_tensor, [-1, size_input])
        else:
            input_tensor_reshaped = input_tensor
        # ---------------------------------------------------------------------------
------------------
        # First FC Layer
        # ---------------------------------------------------------------------------
------------------
        with tf.name_scope('Wx_plus_b_fc_1'):
            preactivate_fc_1 = tf.matmul(input_tensor_reshaped, weights_fc_1) + biase
s_fc_1
            tf.summary.histogram('pre_activations', preactivate_fc_1)

        dropout_act1 = tf.nn.dropout(preactivate_fc_1, keep_prob=dropout_value)
        activations_fc_1 = act(dropout_act1, name='activation_fc_1')

        #tf.summary.histogram('activations_fc_1', activations_fc_1)
        # ---------------------------------------------------------------------------
------------------
        # Second FC Layer
        # ---------------------------------------------------------------------------
------------------
        with tf.name_scope('Wx_plus_b_fc_2'):
            preactivate_fc_2 = tf.matmul(activations_fc_1, weights_fc_2) + biases_fc_
2
            tf.summary.histogram('pre_activations_fc_2', preactivate_fc_2)
        # ---------------------------------------------------------------------------
-------
        # Activation
        # ---------------------------------------------------------------------------
-------
        dropout_act2 = tf.nn.dropout(preactivate_fc_2, keep_prob=dropout_value)
        activations_fc_2 = act(dropout_act2, name='activation_fc_2')
        #
        tf.summary.histogram('activations_fc_2', activations_fc_2)
        # ---------------------------------------------------------------------------
------------------
        # Third FC Layer
        # ---------------------------------------------------------------------------
------------------
        with tf.name_scope('Wx_plus_b_fc_3'):
           preactivate_fc_3 = tf.matmul(activations_fc_2, weights_fc_3) + biases_fc_3
        #      tf.summary.histogram('pre_activations_fc_3', preactivate_fc_3)
        #
        return preactivate_fc_3
        # if act2 is None:
        #     return preactivate_fc_2
        # else:
        #     return activations_fc_2
# ------------------------------------------------------------------------------------
def generate_graph_cnn(shape,num_classes):

    filter_size_1 = 5
    filter_size_2 = 5
    filter_size_3 = 3
    num_filters_1 = 16
    num_filters_2 = 32
    num_filters_3 = 64
    graph_1 = tf.Graph()
    with graph_1.as_default():
```

```
        # Placeholders (Input and Output)
        ph_train = tf.placeholder(tf.float32, shape=(None, shape[0], shape[1], shape[
2]))
        ph_train_labels = tf.placeholder(tf.float32, shape=(None, num_classes))
        keep_prob = tf.placeholder(tf.float32)  # dropout (keep probability)
        # ---------------------------------------------------------------------------
-------
        # Layers definition
        # ---------------------------------------------------------------------------
-------
        # ---------------------------------------------------------------------------
------------------
        # Layer 1
        # ---------------------------------------------------------------------------
------------------
        #LENET
        layer_1 = cnn_layer(ph_train, 'layer_1', filter_size_1, int(shape[2]), num_fi
lters_1, keep_prob, True, act=tf.nn.relu)
        layer_2 = cnn_layer(layer_1, 'layer_2', filter_size_2, int(layer_1.shape[3]),
 num_filters_2, keep_prob, True, act=tf.nn.relu)
        layer_3 = cnn_layer(layer_2, 'layer_3', filter_size_3, int(layer_2.shape[3]),
 num_filters_3, keep_prob, False, act=tf.nn.relu)

        output_nn = full_connect_layer(layer_3, 'Full_Connected_Layer', int(layer_3.s
hape[1] * layer_3.shape[2] * layer_3.shape[3]), 1024, num_classes, keep_prob,False, a
ct=tf.nn.relu, act2=tf.nn.relu)
        # ---------------------------------------------------------------------------
-------
        # Minimization definition
        # ---------------------------------------------------------------------------
-------
        with tf.name_scope('accuracy'):
            # L2 regularitation
            vars = tf.trainable_variables()
            lossL2 = tf.add_n([tf.nn.l2_loss(v) for v in vars if 'weight' in v.name])
 * 0.000001
            loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=outp
ut_nn, labels=ph_train_labels))+lossL2
        tf.summary.scalar('loss', loss)

        labels_pred_softmax = tf.nn.softmax(output_nn)
        correct_pred = tf.equal(tf.argmax(output_nn, 1), tf.argmax(ph_train_labels, 1
))
        accuracy_op = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
        tf.summary.scalar('Accuracy', accuracy_op)
        # ---------------------------------------------------------------------------
-------
        # Trainer
        # ---------------------------------------------------------------------------
-------
        with tf.name_scope('train'):
            optimizer = tf.train.AdamOptimizer(learning_rate=0.001)

        train = optimizer.minimize(loss)
        # ---------------------------------------------------------------------------
-------

    return ph_train, ph_train_labels, output_nn, graph_1, loss, train, keep_prob,accu
racy_op,labels_pred_softmax,layer_3
#-------------------------------------------------------------------------------------
```

```
def get_accuracy(X, Y, session, train, accuracy,labels_pred_softmax,batch_size):

    num_batches = X.shape[0]/batch_size

    final_accuracy = 0
    i_start = 0
    i_end = 0
    accur = 0
    for i in range(np.int(np.ceil(num_batches))):

        if i*batch_size+batch_size < X.shape[0]:
            i_start = i*batch_size
            i_end = i*batch_size+batch_size
        else:
            i_start = i*batch_size
            i_end = X.shape[0]
        x_test= X[i_start:i_end]
        if i == 75:
            pp=0

        x_= X[i_start:i_end]
        y_ = Y[i_start:i_end]
        feed_dict = {ph_train: X[i_start:i_end], ph_train_labels: Y[i_start:i_end], k
eep_prob: 1.0}
        accur, labels_ = session.run([accuracy, labels_pred_softmax], feed_dict=feed_
dict)
        final_accuracy += accur

    return (100.0 * final_accuracy) / np.int(np.ceil(num_batches))
# ------------------------------------------------------------------------------------
-
```

Parameters

In [2]:

```
num_epochs = 14 # num of iterations
num_steps = 100 # each x number of epochs, It will be displayed the loss
batch_size = 128 # Size of the batch
dropout_value = 0.5 # Percentage  to apply in the dropout layer
```

Load data

In [5]:

```
#Load the data augmented
# print('Loading Augmented data ...')
with open('X_train_Augmented_Final_Shuffle.pickle', 'rb') as handle:
     X_train = pickle.load(handle)
# # #
with open('Y_train_Augmented_Final_Shuffle.pickle', 'rb') as handle:
     Y_train = pickle.load(handle)

# print('Loading validation data ...')
with open('X_valid_Normalized.pickle', 'rb') as handle:
     X_valid = pickle.load(handle)
# # #
with open('Y_valid_Normalized.pickle', 'rb') as handle:
     Y_valid = pickle.load(handle)

# print('Loading test data ...')
with open('X_test_Normalized.pickle', 'rb') as handle:
     X_test = pickle.load(handle)
# # #
with open('Y_test_Normalized.pickle', 'rb') as handle:
     Y_test = pickle.load(handle)
```

In [11]:

```
# ---------------------------------------------------------------------------
# Create the graph
# ---------------------------------------------------------------------------
ph_train, ph_train_labels, output_nn, graph_1, loss, train,keep_prob,accuracy_op,labels
_pred_softmax,layer_3 = generate_graph_cnn(image_shape, n_classes)
```

In [ ]:

```
## Training
```

Train, Validate and Test the Model

A validation set can be used to assess how well the model is performing. A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.

In [12]:

```
Debug = 0
tf.logging.set_verbosity(tf.logging.INFO)
cnt = 0
# ---------------------------------------------------------------------------
# Training
# ---------------------------------------------------------------------------
with tf.Session(graph=graph_1) as session:
  tf.global_variables_initializer().run()
  merged_summary = tf.summary.merge_all()
  summary_writer = tf.summary.FileWriter('./Summary', graph_1)
  print('Initialized')
  my_file = Path("./Models/Project2.ckpt.index")
  #if my_file.is_file():
  #   tf.train.Saver().restore(session, "./Models/Project2.ckpt")
```

```
  shape_X = X_train.shape
  if Debug == 0:
      for epoch in range(num_epochs):
          #for iter in range(num_iters):
          for offset in range(0, X_train.shape[0], batch_size):
              end = offset + batch_size
              X_Batch, Y_Batch = X_train[offset:end], Y_train[offset:end]
              #idx = np.random.permutation(X_train.shape[0])[:batch_size]
              feed_dict = {ph_train: X_Batch, ph_train_labels: Y_Batch, keep_prob: dr
opout_value}
              #feed_dict = {ph_train: X_train[idx], ph_train_labels: Y_train[idx], ke
ep_prob: dropout_value}
              _, l, predictions = session.run([train, loss, output_nn], feed_dict=fee
d_dict)

              if offset%(X_train.shape[0]/2) == 0:
                 feed_dict = {ph_train: X_Batch, ph_train_labels: Y_Batch, keep_prob:
 1}
                 #feed_dict = {ph_train: X_train[idx], ph_train_labels: Y_train[idx],
 keep_prob: 1}
                 summary,_, l, predictions,accur = session.run([merged_summary, train
, loss, output_nn, accuracy_op], feed_dict=feed_dict)
                 summary_writer.add_summary(summary, epoch)

          print("EPOCH: ", epoch)
          print("Train accuracy: %.1f%%" % get_accuracy(X_train, Y_train, session, tr
ain, accuracy_op,
                                                              labels_pred_softmax, batch_si
ze))
          print("Valid accuracy: %.1f%%" % get_accuracy(X_valid, Y_valid, session, tr
ain, accuracy_op,
                                                              labels_pred_softmax, batch_si
ze))
          print("Test accuracy: %.1f%%" % get_accuracy(X_test, Y_test, session, train
, accuracy_op,
                                                              labels_pred_softmax, batch_si
ze))
          tf.train.Saver().save(session, "./Models/Project2.ckpt")
      summary_writer.close()
  print("Train accuracy: %.1f%%" % get_accuracy(X_train, Y_train, session, train, acc
uracy_op, labels_pred_softmax, batch_size))
  print("Valid accuracy: %.1f%%" % get_accuracy(X_valid, Y_valid, session, train, acc
uracy_op, labels_pred_softmax, batch_size))
  print("Test accuracy: %.1f%%" % get_accuracy(X_test, Y_test, session, train, accura
cy_op,
                                                              labels_pred_softmax, batch_si
ze))
```

```
Initialized
EPOCH:  0
Train accuracy: 85.0%
Valid accuracy: 88.5%
Test accuracy: 87.0%
EPOCH:  1
Train accuracy: 91.9%
Valid accuracy: 93.8%
Test accuracy: 90.9%
EPOCH:  2
Train accuracy: 94.1%
Valid accuracy: 95.6%
Test accuracy: 92.4%
EPOCH:  3
Train accuracy: 95.3%
Valid accuracy: 95.9%
Test accuracy: 93.0%
EPOCH:  4
Train accuracy: 96.0%
Valid accuracy: 95.7%
Test accuracy: 93.3%
EPOCH:  5
Train accuracy: 96.5%
Valid accuracy: 96.2%
Test accuracy: 93.4%
EPOCH:  6
Train accuracy: 96.7%
Valid accuracy: 96.7%
Test accuracy: 94.1%
EPOCH:  7
Train accuracy: 96.9%
Valid accuracy: 96.6%
Test accuracy: 93.3%
EPOCH:  8
Train accuracy: 97.4%
Valid accuracy: 96.3%
Test accuracy: 93.6%
EPOCH:  9
Train accuracy: 97.7%
Valid accuracy: 97.3%
Test accuracy: 94.0%
EPOCH:  10
Train accuracy: 97.8%
Valid accuracy: 96.6%
Test accuracy: 94.4%
EPOCH:  11
Train accuracy: 97.5%
Valid accuracy: 97.2%
Test accuracy: 94.3%
EPOCH:  12
Train accuracy: 97.8%
Valid accuracy: 96.6%
Test accuracy: 93.9%
EPOCH:  13
Train accuracy: 97.8%
Valid accuracy: 96.6%
Test accuracy: 94.0%
Train accuracy: 97.8%
Valid accuracy: 96.6%
Test accuracy: 94.0%
```

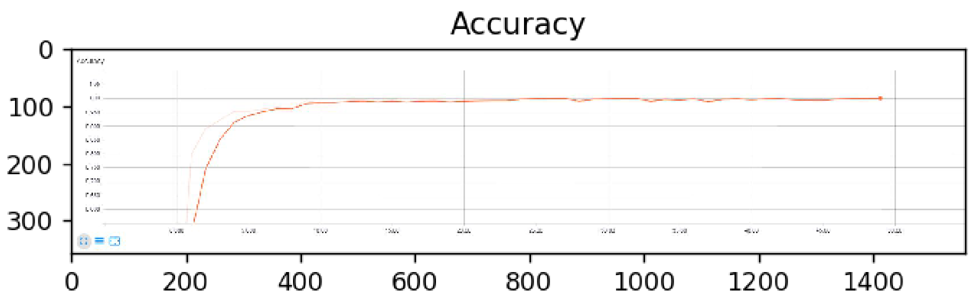
--

Graphs of the Accuracy, Weights distributions and Histograms

In [13]:

```
import matplotlib.pyplot as plt
print("Accuracy")
%matplotlib notebook
im=plt.imread('./images/accuracy_train.JPG')
plt.figure()
plt.title("Accuracy")
plt.imshow(im)
plt.show()
```

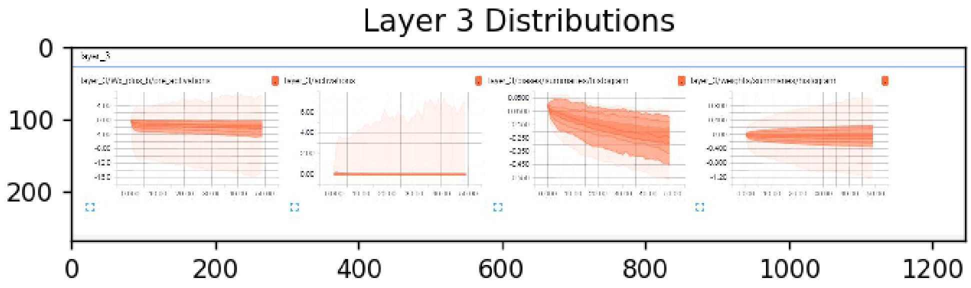
Accuracy



In [14]:

```
import matplotlib.pyplot as plt
print("Layer 3 Distributions")
%matplotlib notebook
im=plt.imread('./images/layer3_summary_distribution.JPG')
plt.figure()
plt.title("Layer 3 Distributions")
plt.imshow(im)
plt.show()
```

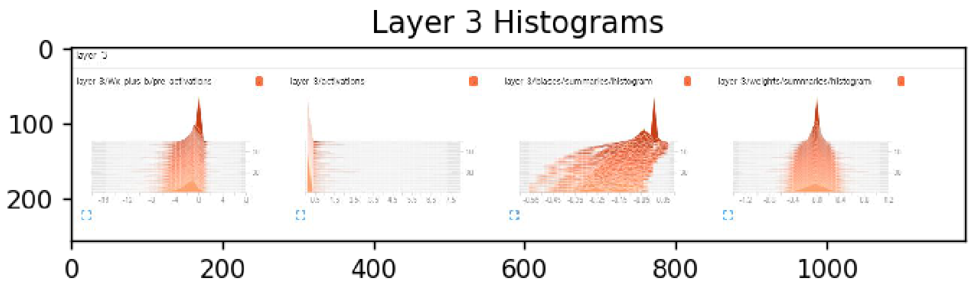
Layer 3 Distributions



In [15]:

```
import matplotlib.pyplot as plt
print("Layer 3 Histograms")
%matplotlib notebook
im=plt.imread('./images/layer3_summary_histogram.JPG')
plt.figure()
plt.title("Layer 3 Histograms")
plt.imshow(im)
plt.show()
```

Layer 3 Histograms



Step 3: Test a Model on New Images

To give yourself more insight into how your model is working, download at least five pictures of German traffic signs from the web and use your model to predict the traffic sign type.

You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

In [16]:

```
import tensorflow as tf
from pathlib import Path
import matplotlib.pyplot as plt
import cv2
import numpy as np
import csv
import matplotlib.gridspec as gridspec
# ---------------------------------------------------------------------------
def normalize_data_test(dataset):
  dataset = dataset.astype(np.float32)

  #grayscale
  dataset = np.uint8(np.sum(dataset / 3, axis=3, keepdims=True))
  for i in range(dataset.shape[0]):
      dataset[i, :, :,0] =  cv2.equalizeHist(dataset[i, :, :,0])

  #normalization
  dataset = (dataset/255)-0.5
  return dataset
# ---------------------------------------------------------------------------
def print_prob_im(im_color,top_5):
    plt.figure(figsize=(5, 1.5))
    gridsp = gridspec.GridSpec(1, 2, width_ratios=[2, 3])
    plt.subplot(gridsp[0])
    plt.imshow(im_color)
    plt.axis('off')
    plt.subplot(gridsp[1])
    list_prob=np.array(top_5[0][0][:])
    print("Prob: ", list_prob[:])
    plt.barh(6 - np.arange(5), list_prob[:], align='center')
    for i_label in range(5):
        plt.text(top_5[0][0][i_label] + .02, 6 - i_label - .25, sign_names[top_5[1][0][
i_label]+1,1])
    plt.axis('off')
    plt.show()
# ---------------------------------------------------------------------------
```

Predict the Sign Type for Each Image

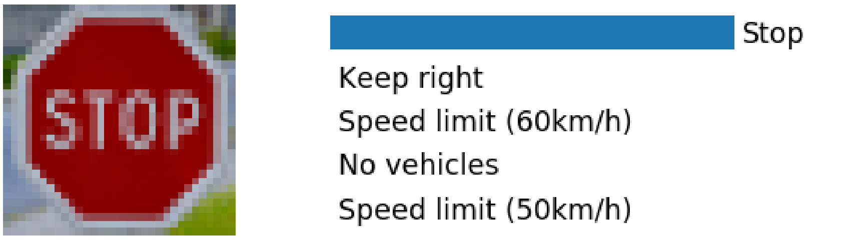
In [3]:

```
### Run the predictions here and use the model to output the prediction for each image.
### Make sure to pre-process the images with the same pre-processing pipeline used earl
ier.
### Feel free to use as many code cells as needed.
```

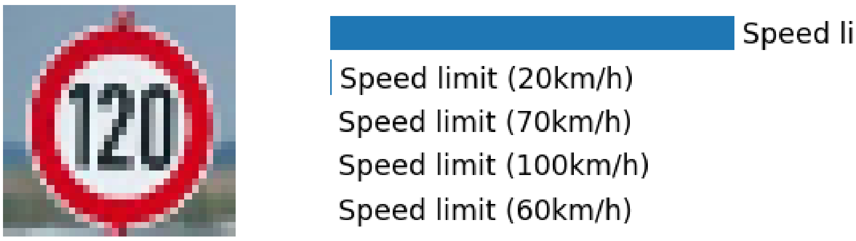
In [18]:

```
with open('./signnames.csv') as f:
    reader = csv.reader(f)
    sign_names = list(reader)
sign_names = np.array(sign_names)
max_score=14

image_shape=[32,32,1]
n_classes=43

image_list =['./images/stop_signal_small.bmp','./images/120_small.bmp','./images/genera
l_caution_small.bmp','./images/Turn_right_ahead_small.bmp','./images/priority_road_smal
l.bmp','./images/no_entry_small.bmp','./images/no_passing_small.bmp','./images/keep_rig
ht_small.bmp']

ph_train, ph_train_labels, output_nn, graph_1, loss, train,keep_prob,accuracy_op,labels
_pred_softmax,layer_3 = generate_graph_cnn(image_shape, n_classes)

with tf.Session(graph=graph_1) as session:
    tf.global_variables_initializer().run()
    merged_summary = tf.summary.merge_all()
    summary_writer = tf.summary.FileWriter('./Summary', graph_1)
    print('Initialized')
    my_file = Path("./Models/Project2.ckpt.index")
    if my_file.is_file():
        tf.train.Saver().restore(session, "./Models/Project2.ckpt")
    # -------------------------------------------------------
    for im_name in image_list:
        im_color = plt.imread(im_name)

        im = np.expand_dims(im_color, axis=0)
        im = normalize_data_test(im)
        # -------------------------------------------------------
        feed_dict = {ph_train: im, keep_prob: 1.0}
        predictions = session.run(labels_pred_softmax, feed_dict=feed_dict)
        top_5 = session.run(tf.nn.top_k(tf.constant(predictions), k=5))

        print_prob_im(im_color, top_5)
```

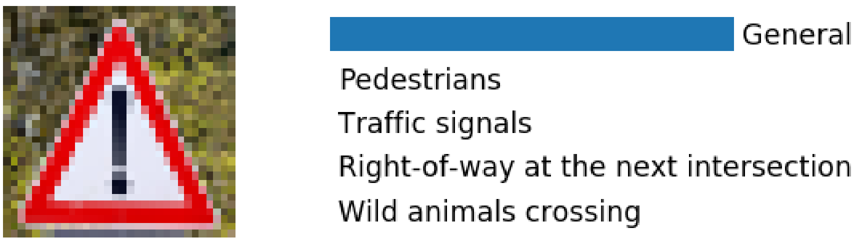
```
Initialized
INFO:tensorflow:Restoring parameters from ./Models/Project2.ckpt
```



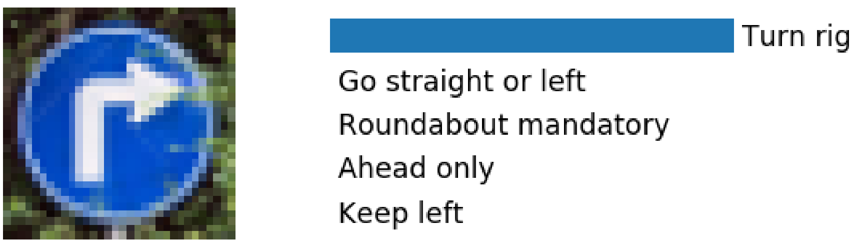

```
Prob:  [  9.99997973e-01   1.50755159e-06   1.99497606e-07   1.53414447e-0
7
   1.31863871e-07]
```



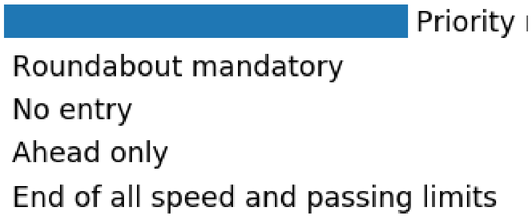

```
Prob:  [  9.98611331e-01   1.38853095e-03   6.74298732e-08   5.60359710e-1
1
   1.23495571e-12]
```



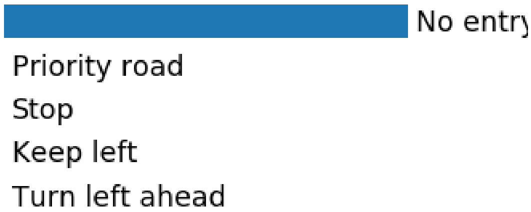

```
Prob:  [  1.00000000e+00   2.00686000e-30   3.20594690e-32   6.18249509e-3
4
   5.39983175e-36]
```



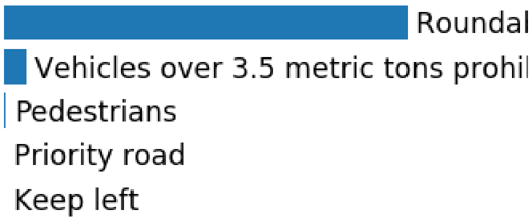

```
Prob:  [  1.00000000e+00   7.62265895e-09   1.94940752e-09   8.24565638e-1
0
   5.14865164e-11]
```



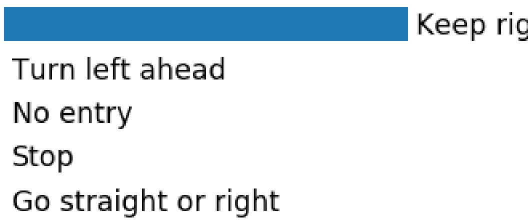

```
Prob:  [  1.00000000e+00   1.71480002e-20   2.65858747e-22   9.82781911e-2
3
   4.77313480e-23]
```




```
Prob:  [  1.00000000e+00   5.97077168e-31   5.17346490e-31   4.25833261e-3
3
   2.92211035e-33]
```




```
Prob:  [  9.47228551e-01   5.17123826e-02   8.54937942e-04   9.99036711e-0
5
   5.03990232e-05]
```




```
Prob:  [  1.00000000e+00   1.41629783e-28   3.59361826e-30   3.31162855e-3
2
   2.86770340e-32]
```

Analyze Performance

Calculate the accuracy for these 5 new images.

For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate on these new images.

The accuracy of the model is 7 out of 8 signs correctly, that's mean that the it is 87.5% accurate in my small dataset. The one that is failling is the "no passing" signal, that is mixing with roundabout mandatory and vehicles over 3.5 Tons prohibited that is quite similar to no passing. In any case the signal has different color that the one in the training dataset and that can give different values on the filters.

Project Writeup

Once you have completed the code implementation, document your results in a project writeup using this template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) as a guide. The writeup can be in a markdown or pdf file.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

Step 4 (Optional): Visualize the Neural Network's State with Test Images

This Section is not required to complete but acts as an additional excersise for understaning the output of a neural network's weights. While neural networks can be a great learning device they are often referred to as a black box. We can understand what the weights of a neural network look like better by plotting their feature maps. After successfully training your neural network you can see what it's feature maps look like by plotting the output of the network's weight layers in response to a test stimuli image. From these plotted feature maps, it's possible to see what characteristics of an image the network finds interesting. For a sign, maybe the inner network feature maps react with high activation to the sign's boundary outline or to the contrast in the sign's painted symbol.

Provided for you below is the function code that allows you to get the visualization output of any tensorflow weight layer you want. The inputs to the function should be a stimuli image, one used during training or a new one you provided, and then the tensorflow variable name that represents the layer's state during the training process, for instance if you wanted to see what the LeNet lab's (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) feature maps looked like for it's second convolutional layer you could enter conv2 as the tf_activation variable.

For an example of what feature map outputs look like, check out NVIDIA's results in their paper End-to-End Deep Learning for Self-Driving Cars (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/) in the section Visualization of internal CNN State. NVIDIA was able to show that their network's inner weights had high activations to road boundary lines by comparing feature maps from an image with a clear path to one without. Try experimenting with a similar test to show that your trained network's weights are looking for interesting features, whether it's looking at differences in feature maps from images with or without a sign, or even what feature maps look like in a trained network vs a completely untrained one on the same sign image.

Combined Image

Your output should look something like this (above)

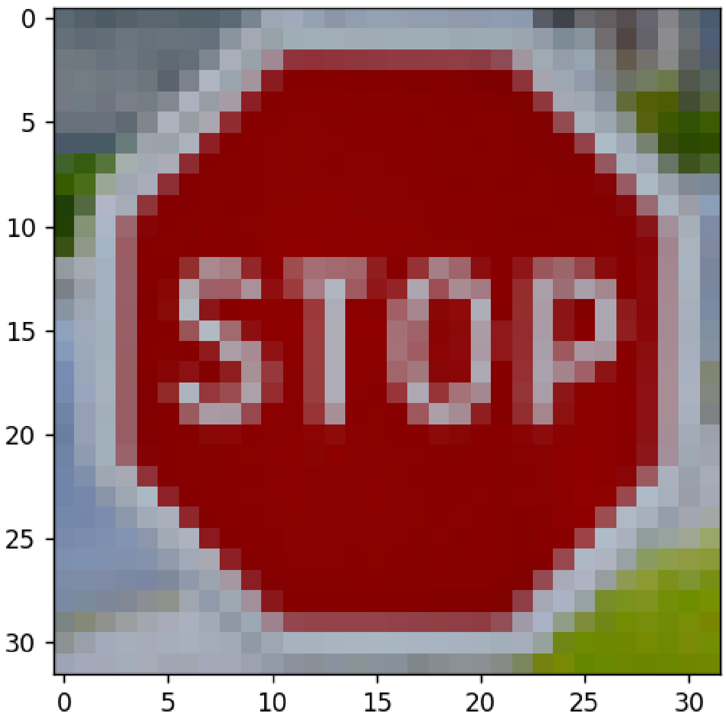
In [19]:

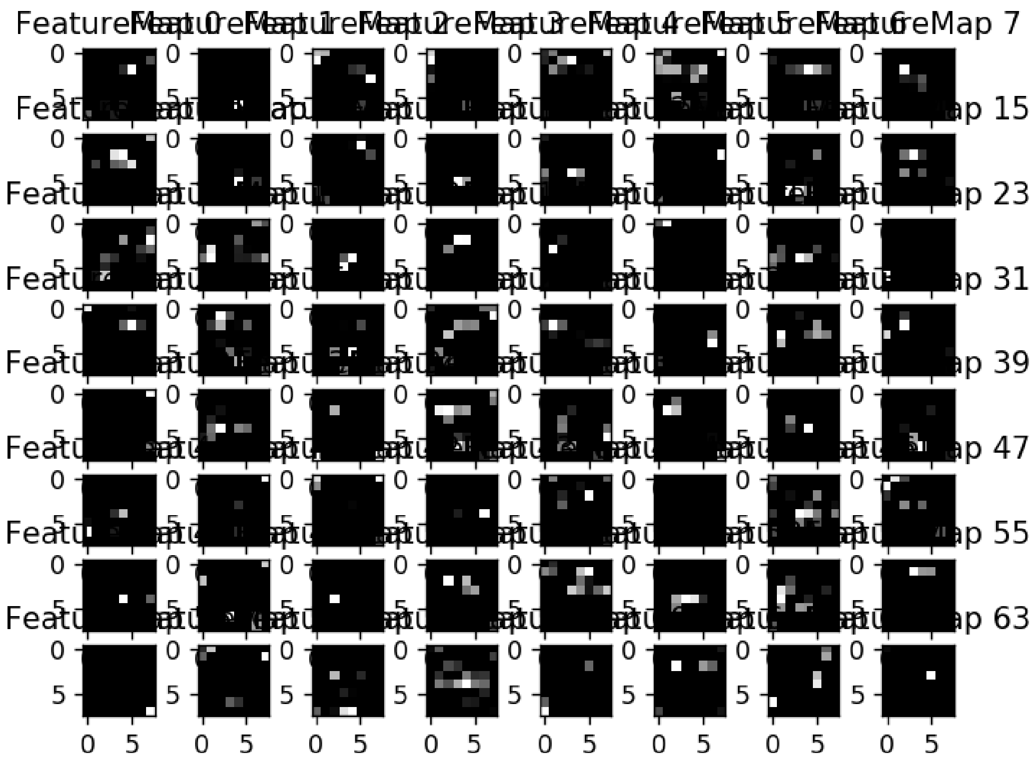
```
import tensorflow as tf
from pathlib import Path
import matplotlib.pyplot as plt
import cv2
import numpy as np
import csv
import matplotlib.gridspec as gridspec
# ------------------------------------------------------------------------------
def normalize_data(dataset):
  dataset = dataset.astype(np.float32)

  #grayscale
  dataset = np.uint8(np.sum(dataset / 3, axis=3, keepdims=True))
  for i in range(dataset.shape[0]):
      dataset[i, :, :,0] =  cv2.equalizeHist(dataset[i, :, :,0])
```

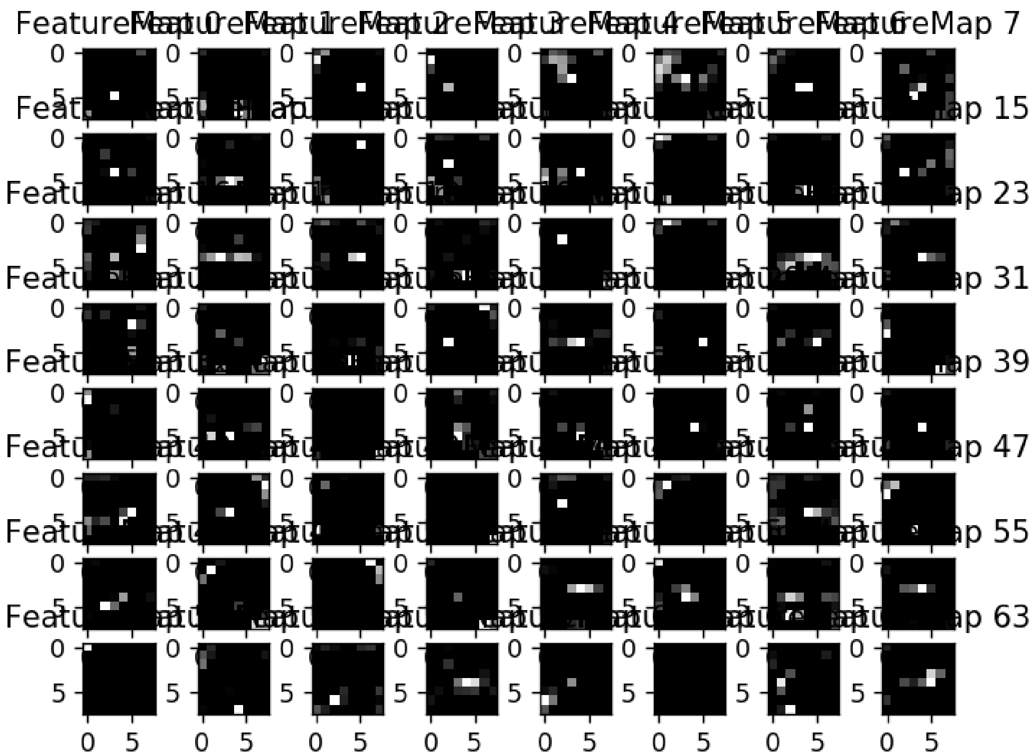
```
  #normalization
  dataset = (dataset/255)-0.5
  return dataset
# ------------------------------------------------------------------------------
def outputFeatureMap(image_input, tf_activation, sess, activation_min=-1, activation_
max=-1 ,plt_num=1):
    # Here make sure to preprocess your image_input in a way your network expects
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow data placeholder v
ariable
    # If you get an error tf_activation is not defined it may be having trouble acces
sing the variable from inside a function


    plt.figure()
    feed_dict_ = {ph_train: image_input, keep_prob: 1.0}

    activation = tf_activation.eval(session=sess, feed_dict=feed_dict_)

    featuremaps = activation.shape[3]

    for featuremap in range(featuremaps):
        plt.subplot(8,8, featuremap+1) # sets the number of feature maps to show on e
ach row and column
        plt.title('FeatureMap ' + str(featuremap)) # displays the feature map number
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin =
activation_min, vmax=activation_max, cmap="gray")
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmax=a
ctivation_max, cmap="gray")
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin=a
ctivation_min, cmap="gray")
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", cmap=
"gray")
    plt.show()

# ------------------------------------------------------------------------------

with open('./signnames.csv') as f:
    reader = csv.reader(f)
    sign_names = list(reader)
sign_names = np.array(sign_names)
max_score=14

image_shape=[32,32,1]
n_classes=43

image_list =['./images/stop_signal_small.bmp','./images/120_small.bmp','./images/gene
ral_caution_small.bmp','./images/Turn_right_ahead_small.bmp','./images/priority_road_
small.bmp','./images/no_entry_small.bmp','./images/no_passing_small.bmp','./images/ke
ep_right_small.bmp']

ph_train, ph_train_labels, output_nn, graph_1, loss, train,keep_prob,accuracy_op,labe
ls_pred_softmax,layer3 = generate_graph_cnn(image_shape, n_classes)

%matplotlib notebook
with tf.Session(graph=graph_1) as session:
```

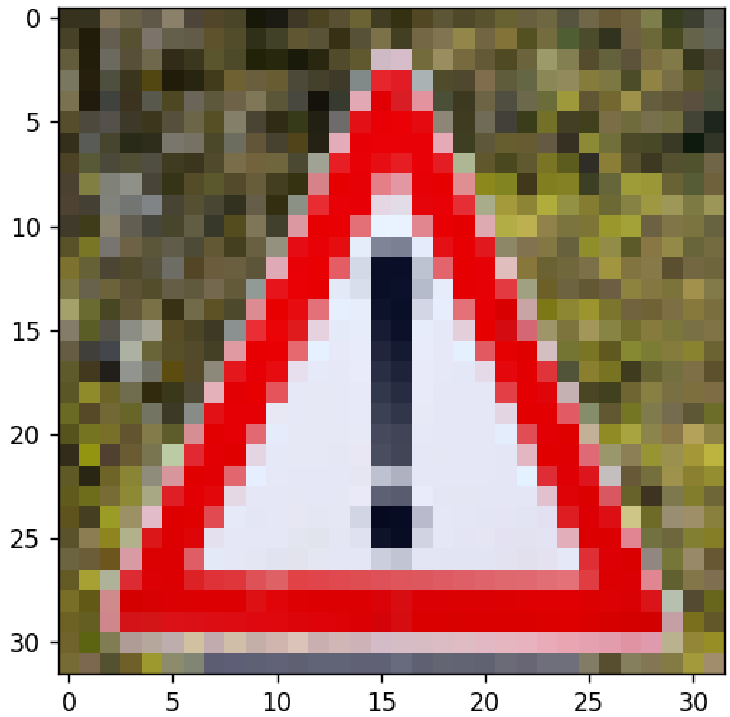
```
    tf.global_variables_initializer().run()
    merged_summary = tf.summary.merge_all()
    summary_writer = tf.summary.FileWriter('./Summary', graph_1)
    print('Initialized')
    my_file = Path("./Models/Project2.ckpt.index")
    if my_file.is_file():
        tf.train.Saver().restore(session, "./Models/Project2.ckpt")
    # -------------------------------------------------------
    for im_name in image_list:
        im_color = plt.imread(im_name)
        plt.figure()
        plt.imshow(im_color)
        plt.show()
        im = np.expand_dims(im_color, axis=0)
        im = normalize_data(im)
        # -------------------------------------------------------
        # Display Layer 3 activations
        outputFeatureMap(im, layer3, session)
```

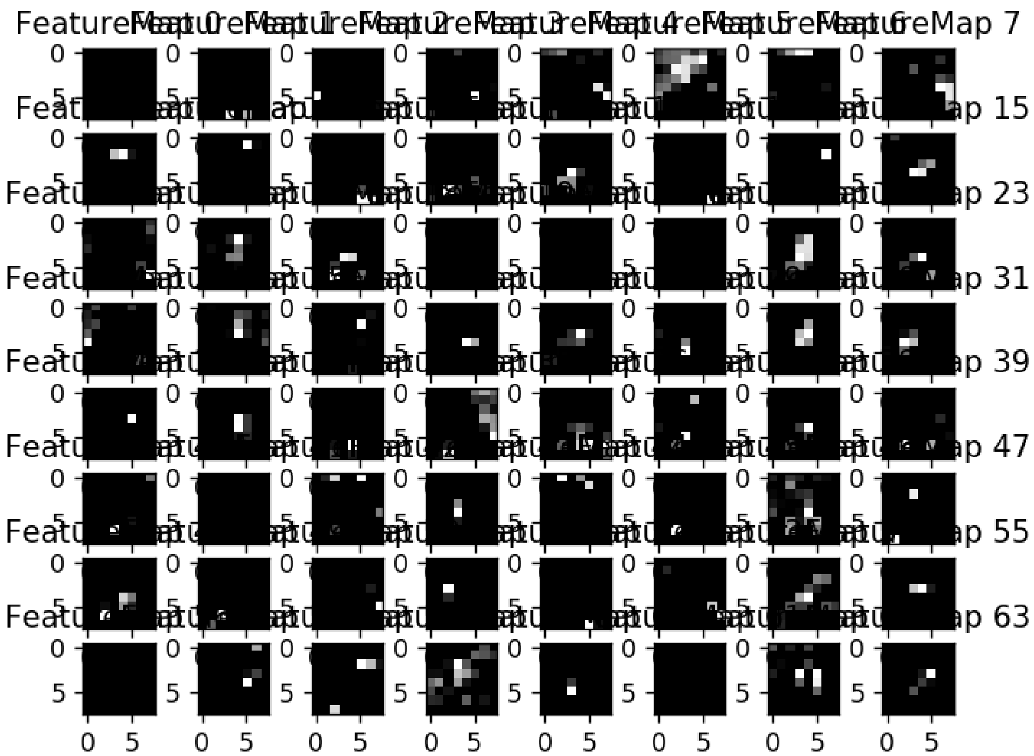
```
Initialized
INFO:tensorflow:Restoring parameters from ./Models/Project2.ckpt
```

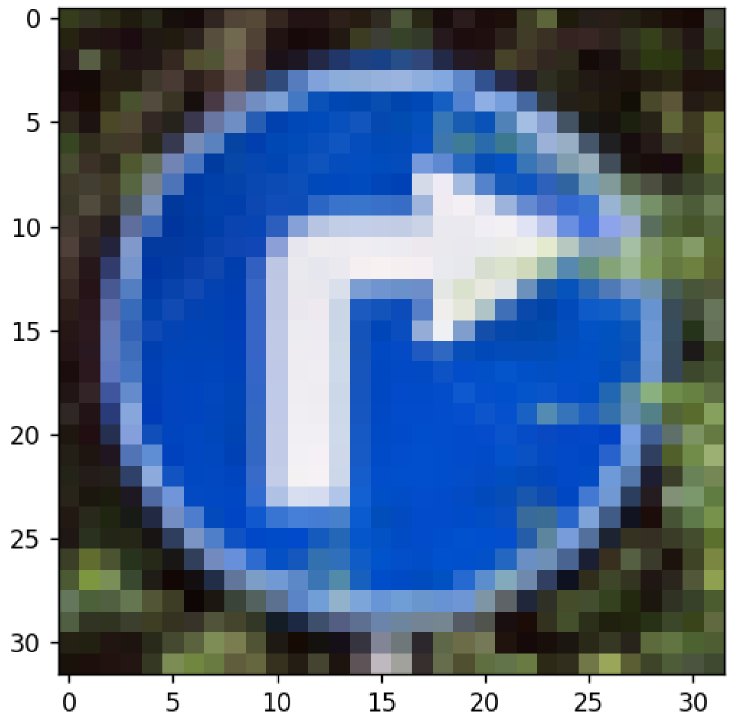





FeatureMap 0
FeatureMap 7
ap 15
ap 23
ap 31
ap 39
ap 47
ap 55
ap 63






0
5
10
15
20
25
30