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 [1]:
```python
# cell 1

# Load pickled data
import pickle

# Fill this in based on where you saved the training and testing data
training_file = 'data/train.p'
validation_file= 'data/valid.p'
testing_file = '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) 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 [2]:
```python
# cell 2

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

# Number of training examples
n_train = X_train.shape[0]

# Number of testing examples.
n_test = X_test.shape[0]
n_valid = X_valid.shape[0]

# What's the shape of an traffic sign image?
image_shape = X_train.shape[1:3]

# How many unique classes/labels there are in the dataset.
n_classes = (np.unique(y_train)).shape[0]

print("Number of training examples =", n_train)
print("Number of validation examples =", n_valid)
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)
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.

In [3]:
```python
# cell 3

### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Visualizations will be shown in the notebook.
%matplotlib inline

plt.rcdefaults()

fig, axes = plt.subplots(nrows=1, ncols=3)

test_data_bin_count = np.bincount(y_train)
test_data_bin_count_nz = np.nonzero(test_data_bin_count)[0]

y_pos = np.arange(len(test_data_bin_count_nz))
x_pos = test_data_bin_count[test_data_bin_count_nz]

axes[0].barh(y_pos, x_pos, align='center', color='blue')
axes[0].set_yticks([y_index for y_index in range(0,len(y_pos),2)], )
axes[0].set_ylabel('Traffic Signs')
axes[0].set_title('Training Data Set')

test_data_bin_count = np.bincount(y_valid)
test_data_bin_count_nz = np.nonzero(test_data_bin_count)[0]

y_pos = np.arange(len(test_data_bin_count_nz))
x_pos = test_data_bin_count[test_data_bin_count_nz]

axes[1].barh(y_pos, x_pos, align='center', color='blue')
axes[1].set_yticks([y_index for y_index in range(0,len(y_pos),2)], )
axes[1].set_xlabel('Number of Examples')
axes[1].set_title('Validation Data Set')

test_data_bin_count = np.bincount(y_test)
test_data_bin_count_nz = np.nonzero(test_data_bin_count)[0]

y_pos = np.arange(len(test_data_bin_count_nz))
x_pos = test_data_bin_count[test_data_bin_count_nz]

axes[2].barh(y_pos, x_pos, align='center', color='blue')
axes[2].set_yticks([y_index for y_index in range(0,len(y_pos),2)], )
axes[2].set_title('Testing Data Set')

plt.show()
fig.savefig('images/data_visualization.jpg')
```
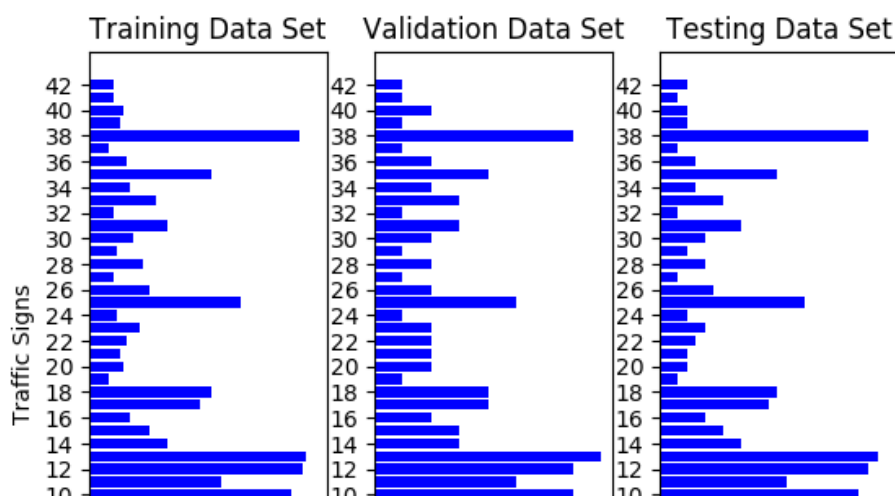
Step 2: Design and Test a Model Architecture

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.

Pre-process the Data Set (normalization, grayscale, etc.)

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

In [4]:
```python
# cell 4

### Preprocess the data here. Preprocessing steps could include normalizati
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import random
import csv

with open('signnames.csv', 'r') as f:
    reader = csv.reader(f)
    signal_names = list(reader)

del signal_names[0]
#print(signal_names)


# Visualizations will be shown in the notebook.
%matplotlib inline
plt.rcdefaults()
plt.figure(figsize=(1,1))
fig, axes = plt.subplots(nrows=1, ncols=2)

index = random.randint(0, len(X_train))
image = X_train[index].squeeze()

axes[0].imshow(image)

print(signal_names[y_train[index]])

#X_train = ( X_train - 128 )
#X_valid = ( X_valid - 128 )
#X_test = ( X_test - 128 )

# Convert to grayscale
X_train = np.mean(X_train, axis=3, keepdims=True)
X_valid = np.mean(X_valid, axis=3, keepdims=True)
X_test = np.mean(X_test, axis=3, keepdims=True)

image = X_train[index].squeeze()

axes[1].imshow(image, cmap="gray")

plt.show()
fig.savefig('images/grayscaling.jpg')
```
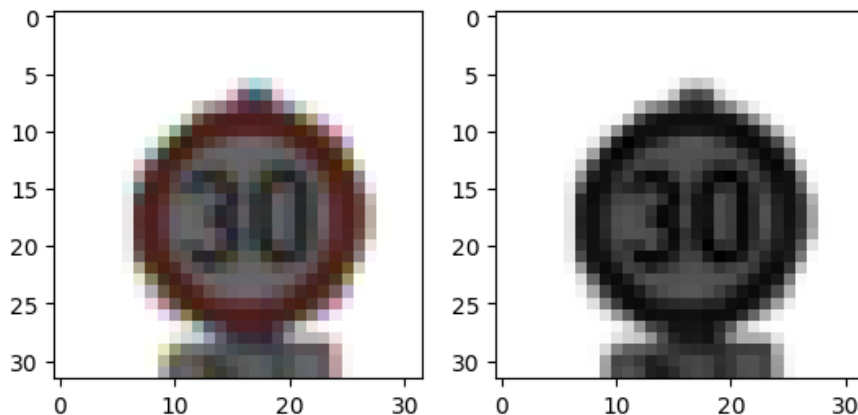
['1', 'Speed limit (30km/h)']

<matplotlib.figure.Figure at 0x7fa0481abd30>

Model Architecture

```
In [5]:  # cell 5

         ### Define your architecture here.
         ### Feel free to use as many code cells as needed.

         import tensorflow as tf

         EPOCHS = 50
         BATCH_SIZE = 64

         MU = 0
         SIGMA = 0.1

         LEARN_RATE = 0.0002
```

```
In [6]:  # cell 6

         # Placeholders for features and labels
         x = tf.placeholder(tf.float32, (None, 32, 32, 1))
         y = tf.placeholder(tf.int32, (None))

         keep_prob = tf.placeholder(tf.float32)

         one_hot_y = tf.one_hot(y, 43)
```

In [7]:
```python
# cell 7

# Construct LeNet architecture
from tensorflow.contrib.layers import flatten

# Layer 1: Convolutional. Input = 32x32x3. Output = 28x28x16.
conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 16), mean = MU, s
conv1_b = tf.Variable(tf.zeros(16))
conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') +

# Activation.
act1 = tf.nn.relu(conv1)

# Pooling. Input = 28x28x16. Output = 14x14x16.
pool1 = tf.nn.max_pool(act1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padd:

# Layer 2: Convolutional. Output = 10x10x32.
conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 16, 32), mean = MU, 
conv2_b = tf.Variable(tf.zeros(32))
conv2   = tf.nn.conv2d(pool1, conv2_W, strides=[1, 1, 1, 1], padding='VALID

# Activation.
act2 = tf.nn.relu(conv2)

# Pooling. Input = 10x10x32. Output = 5x5x32.
pool2 = tf.nn.max_pool(act2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padd:

# Dropout. Input = 5x5x32. Output = 5x5x32.
dropout3 = tf.nn.dropout(pool2,keep_prob)

# Flatten. Input = 5x5x32. Output = 800.
fc0   = flatten(dropout3)

# Layer 3: Fully Connected. Input = 800. Output = 400.
fc1_W = tf.Variable(tf.truncated_normal(shape=(800, 400), mean = MU, stddev
fc1_b = tf.Variable(tf.zeros(400))
fc1   = tf.matmul(fc0, fc1_W) + fc1_b

# Activation.
fc1   = tf.nn.relu(fc1)

# Layer 4: Fully Connected. Input = 400. Output = 200.
fc2_W  = tf.Variable(tf.truncated_normal(shape=(400, 200), mean = MU, stddev
fc2_b  = tf.Variable(tf.zeros(200))
fc2    = tf.matmul(fc1, fc2_W) + fc2_b

# Activation.
fc2    = tf.nn.relu(fc2)

# Layer 5: Fully Connected. Input = 200. Output = 120.
fc3_W  = tf.Variable(tf.truncated_normal(shape=(200, 120), mean = MU, stddev
fc3_b  = tf.Variable(tf.zeros(120))
fc3 = tf.matmul(fc2, fc3_W) + fc3_b

# Activation.
fc3    = tf.nn.relu(fc3)

# Layer 6: Fully Connected. Input = 120. Output = 43.
fc4_W  = tf.Variable(tf.truncated_normal(shape=(120, 43), mean = MU, stddev
fc4_b  = tf.Variable(tf.zeros(43))
logits = tf.matmul(fc3, fc4_W) + fc4_b
```

In [8]:
```python
# cell 8

# Training pipeline
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, l
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = LEARN_RATE)
training_operation = optimizer.minimize(loss_operation)
```

In [9]:
```python
# cell 9

# Model evaluation pipeline
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1)
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)

def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: b
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

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 [10]:
```python
# cell 10

### Train your model here.
### Calculate and report the accuracy on the training and validation set.
### Once a final model architecture is selected,
### the accuracy on the test set should be calculated and reported as well.
### Feel free to use as many code cells as needed.
from sklearn.utils import shuffle

saver = tf.train.Saver()

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    print("Training...")
    print()
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x: batch_x, y: batch_y,

        validation_accuracy = evaluate(X_valid, y_valid)
        training_accuracy = evaluate(X_train, y_train)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print("Training Accuracy   = {:.3f}".format(training_accuracy))
        print()

    saver.save(sess, './lenet')
    print("Model saved")
```

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.099
Training Accuracy   = 0.100

EPOCH 2 ...
Validation Accuracy = 0.180
Training Accuracy   = 0.204

EPOCH 3 ...
Validation Accuracy = 0.302
Training Accuracy   = 0.359

EPOCH 4 ...
Validation Accuracy = 0.422
Training Accuracy   = 0.478

EPOCH 5 ...
Validation Accuracy = 0.472
```

In [20]:
```python
# cell 11

# Evaluate model accuracy on the test set
saver = tf.train.Saver()

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(X_test, y_test)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

Test Accuracy = 0.946

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.

Load and Output the Images

In [21]:
```python
# cell 12

### Load the images and plot them here.
### Feel free to use as many code cells as needed.
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

files_web = ['web/web1_.jpeg','web/web2_.jpeg','web/web3_.jpeg','web/web4_..
y_web = [3,12,11,2,14]

plt.rcdefaults()

fig, axes = plt.subplots(nrows=2, ncols=5)

x_web = None

for i in range(len(files_web)):
    image = mpimg.imread(files_web[i])
    axes[0][i].imshow(image)

    image = np.mean(image, axis=2, keepdims=True)
    axes[1][i].imshow(image.squeeze(), cmap="gray")

    imshape = (1,)+image.shape

    if x_web == None:
        x_web = image.reshape(imshape)
    else:
        x_web = np.concatenate((x_web, image.reshape(imshape)), axis=0)
```

/home/isharaka/miniconda3/envs/IntroToTensorFlow/lib/python3.6/site-packag
es/ipykernel/__main__.py:24: FutureWarning: comparison to `None` will resu
lt in an elementwise object comparison in the future.

Predict the Sign Type for Each Image

In [22]:
```python
# cell 13

### Run the predictions here and use the model to output the prediction for
### Make sure to pre-process the images with the same pre-processing pipeli
### Feel free to use as many code cells as needed.


with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    prediction = sess.run(tf.argmax(logits, 1),feed_dict={x: x_web, y: y_wel
    print (prediction)
```

[3 12 11 2 14]

Analyze Performance

In [23]:
```python
# cell 14

### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    accuracy = sess.run(accuracy_operation,feed_dict={x: x_web, y: y_web, ke
    print (accuracy)
```

1.0

Output Top 5 Softmax Probabilities For Each Image Found on the Web

For each of the new images, print out the model's softmax probabilities to show the **certainty** of the model's predictions (limit the output to the top 5 probabilities for each image). `tf.nn.top_k` (https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) could prove helpful here.

The example below demonstrates how tf.nn.top_k can be used to find the top k predictions for each image.

`tf.nn.top_k` will return the values and indices (class ids) of the top k predictions. So if k=3, for each sign, it'll return the 3 largest probabilities (out of a possible 43) and the correspoding class ids.

Take this numpy array as an example. The values in the array represent predictions. The array contains softmax probabilities for five candidate images with six possible classes. `tk.nn.top_k` is used to choose the three classes with the highest probability:

```
# (5, 6) array
a = np.array([[ 0.24879643,  0.07032244,  0.12641572,  0.34763842,
0.07893497,
        0.12789202],
     [ 0.28086119,  0.27569815,  0.08594638,  0.0178669 ,  0.18063
401,
        0.15899337],
     [ 0.26076848,  0.23664738,  0.08020603,  0.07001922,  0.11343
71 ,
        0.23892179],
     [ 0.11943333,  0.29198961,  0.02605103,  0.26234032,  0.13513
48 ,
        0.16505091],
     [ 0.09561176,  0.34396535,  0.0643941 ,  0.16240774,  0.24206
137,
        0.09155967]])
```

Running it through `sess.run(tf.nn.top_k(tf.constant(a), k=3))` produces:

```
TopKV2(values=array([[ 0.34763842,  0.24879643,  0.12789202],
     [ 0.28086119,  0.27569815,  0.18063401],
     [ 0.26076848,  0.23892179,  0.23664738],
     [ 0.29198961,  0.26234032,  0.16505091],
     [ 0.34396535,  0.24206137,  0.16240774]]), indices=array([[3,
0, 5],
     [0, 1, 4],
     [0, 5, 1],
     [1, 3, 5],
     [1, 4, 3]], dtype=int32))
```

Looking just at the first row we get `[ 0.34763842,  0.24879643,  0.12789202]`, you can confirm these are the 3 largest probabilities in a. You'll also notice `[3, 0, 5]` are the corresponding indices.

In [24]:
```python
# cell 15

### Print out the top five softmax probabilities for the predictions on the
### Feel free to use as many code cells as needed.

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    accuracy = sess.run(tf.nn.top_k(logits, k=5),feed_dict={x: x_web, keep_
    print (accuracy)
```

```
TopKV2(values=array([[ 37.11714554,  27.79857445,  22.40261841,  21.500734
33,
          16.06358147],
       [ 30.95289803,  22.71965408,  14.15060902,  12.31142044,
          10.84618664],
       [ 77.54013824,  51.27370834,  41.99422455,  34.29644394,
          31.51921463],
       [ 49.16646194,  33.07032776,  32.0014534 ,  30.65810394,
          28.96285057],
       [ 28.94649124,  14.52554035,   9.29852676,   7.65595484,
           4.84923983]], dtype=float32), indices=array([[ 3,  2,  5,  1,  4
],
       [12, 40, 38, 15,  8],
       [11, 28, 30, 23, 27],
       [ 2,  3,  5,  1,  7],
       [14, 38, 13, 40, 34]], dtype=int32))
```

In [25]:
```python
# cell 16

for i in range(len(accuracy[0])):
    labels = accuracy[1][i]
    sm_prob = accuracy[0][i]/sum(accuracy[0][i])

    print("Expected prediction:%3d  %s" % (y_web[i],signal_names[y_web[i]][
    for j in range(len(labels)):
        print("Prediction:%3d   Softmax Prob.: %1.2f   %s" % (labels[j],sm_

    print(' ')
```

```
Expected prediction:  3  Speed limit (60km/h)
Prediction:  3   Softmax Prob.: 0.30   Speed limit (60km/h)
Prediction:  2   Softmax Prob.: 0.22   Speed limit (50km/h)
Prediction:  5   Softmax Prob.: 0.18   Speed limit (80km/h)
Prediction:  1   Softmax Prob.: 0.17   Speed limit (30km/h)
Prediction:  4   Softmax Prob.: 0.13   Speed limit (70km/h)

Expected prediction: 12  Priority road
Prediction: 12   Softmax Prob.: 0.34   Priority road
Prediction: 40   Softmax Prob.: 0.25   Roundabout mandatory
Prediction: 38   Softmax Prob.: 0.16   Keep right
Prediction: 15   Softmax Prob.: 0.14   No vehicles
Prediction:  8   Softmax Prob.: 0.12   Speed limit (120km/h)

Expected prediction: 11  Right-of-way at the next intersection
Prediction: 11   Softmax Prob.: 0.33   Right-of-way at the next intersecti
on
Prediction: 28   Softmax Prob.: 0.22   Children crossing
Prediction: 30   Softmax Prob.: 0.18   Beware of ice/snow
Prediction: 23   Softmax Prob.: 0.14   Slippery road
Prediction: 27   Softmax Prob.: 0.13   Pedestrians

Expected prediction:  2  Speed limit (50km/h)
Prediction:  2   Softmax Prob.: 0.28   Speed limit (50km/h)
Prediction:  3   Softmax Prob.: 0.19   Speed limit (60km/h)
Prediction:  5   Softmax Prob.: 0.18   Speed limit (80km/h)
Prediction:  1   Softmax Prob.: 0.18   Speed limit (30km/h)
Prediction:  7   Softmax Prob.: 0.17   Speed limit (100km/h)

Expected prediction: 14  Stop
Prediction: 14   Softmax Prob.: 0.44   Stop
Prediction: 38   Softmax Prob.: 0.22   Keep right
Prediction: 13   Softmax Prob.: 0.14   Yield
Prediction: 40   Softmax Prob.: 0.12   Roundabout mandatory
Prediction: 34   Softmax Prob.: 0.07   Turn left ahead
```

In [39]:
```python
# cell 17

plt.rcdefaults()

fig, axes = plt.subplots(nrows=5, ncols=2)

fig.tight_layout()

for i in range(5):
    y_pos = accuracy[1][i]
    x_pos = ( accuracy[0][i] / sum(accuracy[0][i]) )

    people = accuracy[1][i]
    y_pos = np.arange(len(accuracy[1][i]))
    performance = accuracy[0][i]

    axes[i][0].barh(y_pos, x_pos, align='center',color='green')
    axes[i][0].set_yticks(y_pos)
    axes[i][0].set_yticklabels(accuracy[1][i])
    axes[i][0].invert_yaxis()  # labels read top-to-bottom
    axes[i][0].set_title('Prediction: '+str(accuracy[1][i][0]))

    image = mpimg.imread(files_web[i])
    axes[i][1].imshow(image)
    axes[i][1].set_title('Expected prediction: '+str(y_web[i]))

axes[2][0].set_ylabel('Predictions with top 5 softmax prob.')
axes[4][0].set_xlabel('Softmax probability')

plt.show()
fig.savefig('images/softmax_visualization.jpg')
```
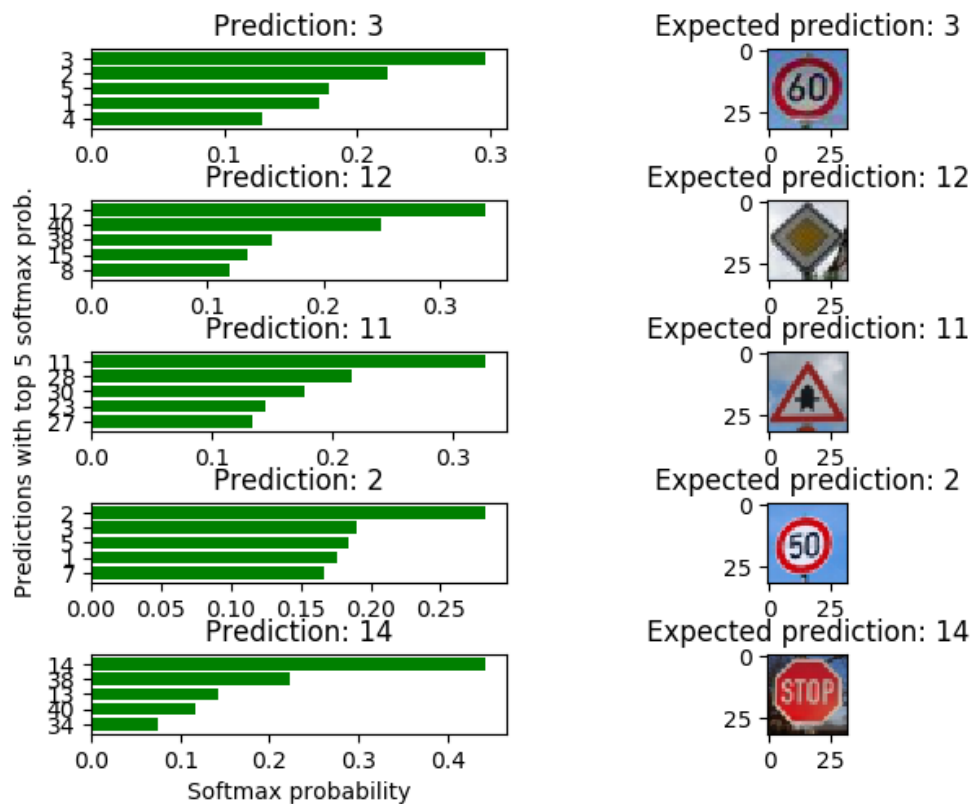
Step 4: Visualize the Neural Network's State with Test Images

This Section is not required to complete but acts as an additional excersise for understanding 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.
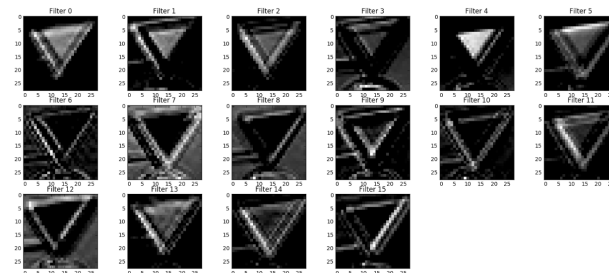
Your output should look something like this (above)

In [27]:
```python
# cell 18

### Visualize your network's feature maps here.
### Feel free to use as many code cells as needed.

# image_input: the test image being fed into the network to produce the fea
# tf_activation: should be a tf variable name used during your training pro
# activation_min/max: can be used to view the activation contrast in more d
# plt_num: used to plot out multiple different weight feature map sets on t

def outputFeatureMap(image_input, tf_activation, activation_min=-1, activat
    # Here make sure to preprocess your image_input in a way your network e
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow data pla
    # If you get an error tf_activation is not defined it maybe having trou
    activation = tf_activation.eval(session=sess,feed_dict={x : image_input
    featuremaps = activation.shape[3]
    plt.figure(plt_num, figsize=(15,15))
    for featuremap in range(featuremaps):
        plt.subplot(6,8, featuremap+1) # sets the number of feature maps to
        plt.title('FeatureMap ' + str(featuremap)) # displays the feature m
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="neares
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="neares
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="neares
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation="neares
```
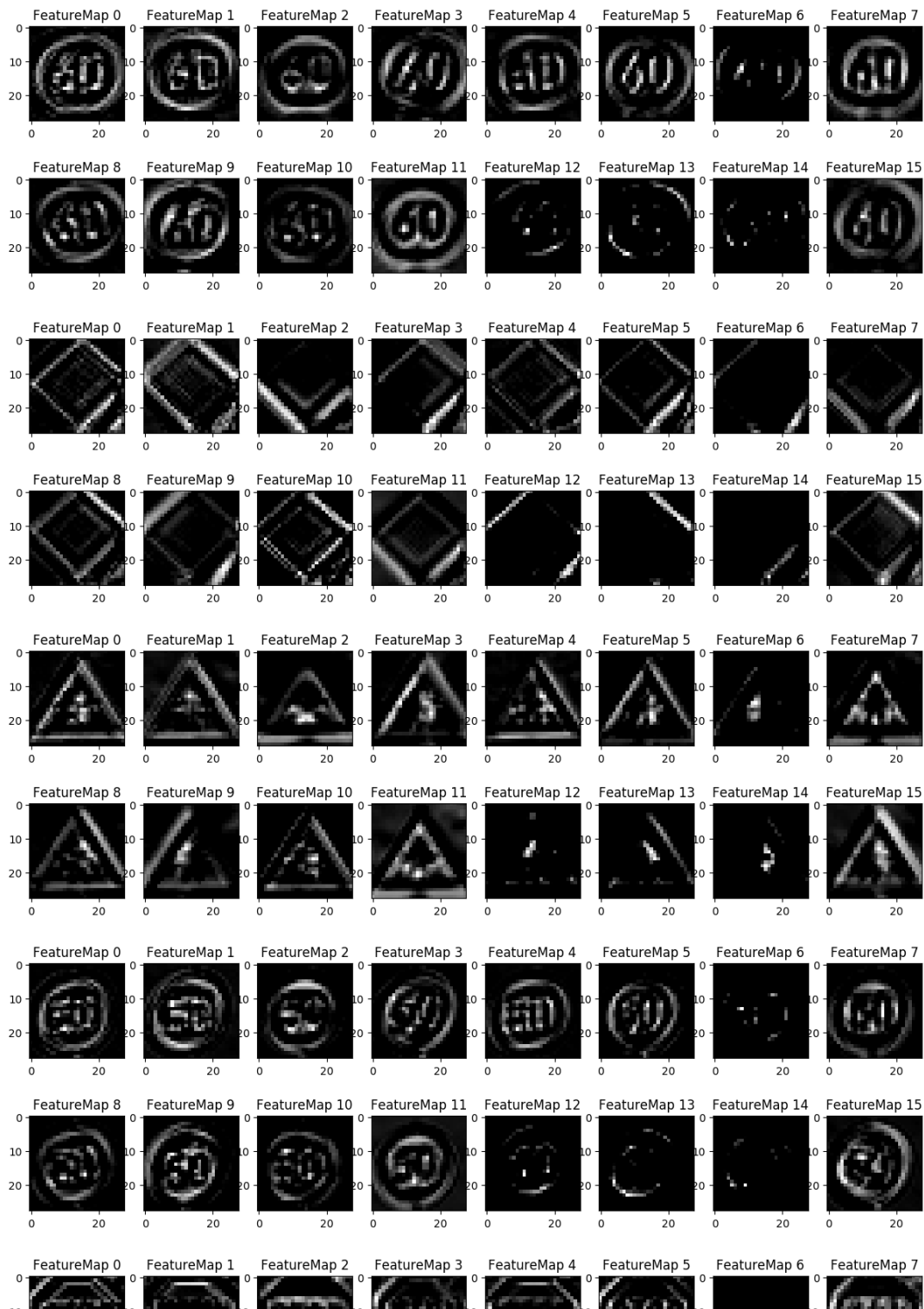
In [40]:
```python
# cell 19

plt.rcdefaults()

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    for i in range(len(x_web)):
        outputFeatureMap(x_web[i].reshape(1,32,32,1),act1,plt_num=2*i)
        #outputFeatureMap(x_web[i].reshape(1,32,32,1),act2,plt_num=2*i+1)

plt.show()
```

Question 9

Discuss how you used the visual output of your trained network's feature maps to show that it had learned to look for interesting characteristics in traffic sign images

Answer:

> **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.

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.