Self-Driving Car Engineer Nanodegree ¶

[https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project)](https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project)

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)](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)](https://review.udacity.com/#!/rubrics/481/view) for this project.

The [rubric (https://review.udacity.com/#!/rubrics/481/view)](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]:
```
import pickle
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Visualizations will be shown in the notebook.
%matplotlib inline
```

In [2]:
```
# Load pickled data

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

DATA_DIR = "../../data/traffic-signs/"
training_file = DATA_DIR + "train.p"
validation_file= DATA_DIR + "valid.p"
testing_file = DATA_DIR + "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_val, y_val = 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 [3]:

```python
# Summary of data

n_train = len(X_train)
n_val = len(X_val)
n_test = len(X_test)
image_shape = X_train.shape[1:]
n_classes = len(np.unique(y_train))

print("Number of training examples =", n_train)
print("Number of examples: training: %d, validation: %d, testing: %d" %
print("Image data shape: ", image_shape)
print("Number of classes: ", n_classes)

classes, counts = np.unique(y_train, return_counts=True)
signnames_df = pd.read_csv("./signnames.csv")
signnames={int(row["ClassId"]): row["SignName"] for _, row in signnames_

print(pd.DataFrame({"class":classes, "count":counts, "name": signnames_(
```

```
Number of training examples = 34799
Number of examples: training: 34799, validation: 4410, testing: 12630
Image data shape:  (32, 32, 3)
Number of classes:  43
    class  count                                                name
0       0    180                                Speed limit (20km/h)
1       1   1980                                Speed limit (30km/h)
2       2   2010                                Speed limit (50km/h)
3       3   1260                                Speed limit (60km/h)
4       4   1770                                Speed limit (70km/h)
5       5   1650                                Speed limit (80km/h)
6       6    360                         End of speed limit (80km/h)
7       7   1290                               Speed limit (100km/h)
8       8   1260                               Speed limit (120km/h)
9       9   1320                                          No passing
10     10   1800        No passing for vehicles over 3.5 metric tons
11     11   1170               Right-of-way at the next intersection
12     12   1890                                       Priority road
13     13   1920                                               Yield
14     14    690                                                Stop
15     15    540                                         No vehicles
16     16    360            Vehicles over 3.5 metric tons prohibited
17     17    990                                            No entry
18     18   1080                                     General caution
19     19    180                         Dangerous curve to the left
20     20    300                        Dangerous curve to the right
21     21    270                                        Double curve
22     22    330                                          Bumpy road
23     23    450                                       Slippery road
24     24    240                           Road narrows on the right
25     25   1350                                           Road work
26     26    540                                     Traffic signals
27     27    210                                         Pedestrians
28     28    480                                   Children crossing
29     29    240                                   Bicycles crossing
30     30    390                                  Beware of ice/snow
31     31    690                               Wild animals crossing
32     32    210                 End of all speed and passing limits
```

```
33      33    599                                  Turn right ahead
34      34    360                                   Turn left ahead
35      35   1080                                        Ahead only
36      36    330                              Go straight or right
37      37    180                               Go straight or left
38      38   1860                                        Keep right
39      39    270                                         Keep left
40      40    300                              Roundabout mandatory
41      41    210                                 End of no passing
42      42    210  End of no passing by vehicles over 3.5 metric ...
```

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/)](http://matplotlib.org/) [examples (http://matplotlib.org/examples/index.html)](http://matplotlib.org/examples/index.html) and [gallery (http://matplotlib.org/gallery.html)](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 [4]:
```
# Plot example traffic signs
cols = 3
rows = len(classes)
fig, axes = plt.subplots(rows, cols, figsize=(4*cols, 4*rows))
np.random.seed(1)
for i, class_id in enumerate(classes[:rows]):
  indices = np.where(y_train==class_id)[0]
  indices = np.random.choice(indices, cols, replace=False)
  for j, idx in enumerate(indices):
    axes[i, j].imshow(X_train[idx,:,:,:])
    axes[i, j].set_xticks([])
    axes[i, j].set_yticks([])
  axes[i, 0].set_title("%d: %s" % (class_id, signnames[class_id]))
```

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)](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.)

Minimally, the image data should be normalized so that the data has mean zero and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize the data and can be used in this project.

Other pre-processing steps are optional. You can try different techniques to see if it improves performance.

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

In [5]:

```
# Preprocess: normalize, grayscale?
def normalize(X):
  return (X - 128.0) / 128.0

X_train = normalize(X_train)
X_val = normalize(X_val)
X_test = normalize(X_test)
#print(X_test.mean())
```

Model Architecture

In [6]:
```
# Model
import tensorflow as tf

def TrafficSignModel(X, n_classes, dropout, is_training):
  # Conv1: input: 32x32x3, output: 28x28x6
  conv1 = conv2d(X, ksize=5, stride=1, in_channels=3, out_channels=6)
  conv1 = tf.nn.relu(conv1)
  conv1 = max_pool(conv1, ksize=2, stride=2)
  conv1 = tf.layers.dropout(conv1, rate=dropout, training=is_training)

  # Conv2: input: 14x14x6, output: 10x10x16
  conv2 = conv2d(conv1, ksize=5, stride=1, in_channels=6, out_channels=
  conv2 = tf.nn.relu(conv2)
  conv2 = max_pool(conv2, ksize=2, stride=2)
  conv2 = tf.layers.dropout(conv2, rate=dropout, training=is_training)

  # Input: 5x5x16, output: 400
  fc0 = tf.layers.flatten(conv2)

  # FC1: input: 400, output: 120
  fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean=0, std
  fc1_b = tf.Variable(tf.zeros(120))
  fc1 = tf.matmul(fc0, fc1_W) + fc1_b
  fc1 = tf.nn.relu(fc1)
  fc1 = tf.layers.dropout(fc1, rate=dropout, training=is_training)

  # FC2:
  fc2_W = tf.Variable(tf.truncated_normal(shape=(120, 84), mean=0, stdd
  fc2_b = tf.Variable(tf.zeros(84))
  fc2 = tf.matmul(fc1, fc2_W) + fc2_b
  fc2 = tf.nn.relu(fc2)

  fc3_W = tf.Variable(tf.truncated_normal(shape=(84, n_classes), mean=0
  fc3_b = tf.Variable(tf.zeros(n_classes))
  logits = tf.matmul(fc2, fc3_W) + fc3_b

  return logits


def conv2d(input, ksize, stride, in_channels, out_channels, padding="VA
  W = tf.Variable(tf.truncated_normal(shape=(ksize, ksize, in_channels,
  b = tf.Variable(tf.zeros(out_channels))
  conv = tf.nn.conv2d(input, W, strides=[stride, stride, stride,stride]
  return conv

def max_pool(input, ksize, stride, padding="VALID"):
  return tf.nn.max_pool(input, ksize=[1, ksize, ksize, 1], strides=[1,
```

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 [7]:

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

def train(n_classes, X_train, y_train, X_val, y_val,
          epochs=1, learning_rate=0.001, batch_size=64, dropout=0.0, see
  tf.set_random_seed(seed)
  X = tf.placeholder(tf.float32, (None, 32, 32, 3))
  y = tf.placeholder(tf.int32, (None))
  y_one_hot = tf.one_hot(y, n_classes)
  is_training = tf.placeholder_with_default(False, shape=())

  logits = TrafficSignModel(X, n_classes, dropout, is_training)
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_one_h
  cost = tf.reduce_mean(cross_entropy)
  optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimi
  prediction_op = tf.cast(tf.argmax(logits, 1), tf.int32)
  correct_op = tf.equal(prediction_op, y)
  accuracy_op = tf.reduce_mean(tf.cast(correct_op, tf.float32))
  saver = tf.train.Saver()

  metrics={"train_loss": [], "val_loss":[], "cost":[]}
  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    n_examples = len(X_train)
    for i in range(epochs):
      X_train, y_train = shuffle(X_train, y_train)
      for offset in range(0, n_examples, batch_size):
        end = offset + batch_size
        batch_X, batch_y = X_train[offset:end], y_train[offset:end]
        sess.run(optimizer, feed_dict={X: batch_X, y: batch_y, is_traini
      train_accuracy = accuracy_op.eval({X: X_train, y: y_train})
      validation_accuracy = accuracy_op.eval({X: X_val, y: y_val})
      metrics["train_loss"].append(1.0 - train_accuracy)
      metrics["val_loss"].append(1.0 - validation_accuracy)
      print("Epoch %2d: train accuracy: %.3f, validation accuracy: %.3f
            train_accuracy, validation_accuracy))
    saver.save(sess, DATA_DIR + "traffic_signs_net")
    print("Model saved.")
  vars = {
    "X": X,
    "y": y,
    "logits": logits,
    "prediction_op": prediction_op,
    "accuracy_op": accuracy_op
  }
  return vars, metrics

def evaluate(vars, X_data, y_data, batch_size=64):
  n_examples = len(X_data)
  total_accuracy = 0.0
  sess = tf.get_default_session()
```

```
  for offset in range(0, n_examples, batch_size):
    batch_X, batch_y = X_data[offset : offset + batch_size], y_data[offs
    accuracy = sess.run(vars["accuracy_op"], feed_dict={vars["X"]: batcl
    total_accuracy += accuracy * len(batch_X)
  total_accuracy /= n_examples
  return total_accuracy

epochs = 20
vars, metrics = train(n_classes, X_train, y_train, X_val, y_val, epochs

plt.plot(range(epochs), metrics["train_loss"], label="train")
plt.plot(range(epochs), metrics["val_loss"], label="validation")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()


def test_model(vars, X_test, y_test):
  saver = tf.train.Saver()
  with tf.Session() as sess:
    saver.restore(sess, DATA_DIR + "traffic_signs_net")
    test_accuracy = evaluate(vars, X_test, y_test)
    print("Test accuracy: %.3f\n" % test_accuracy)

test_model(vars, X_test, y_test)
```

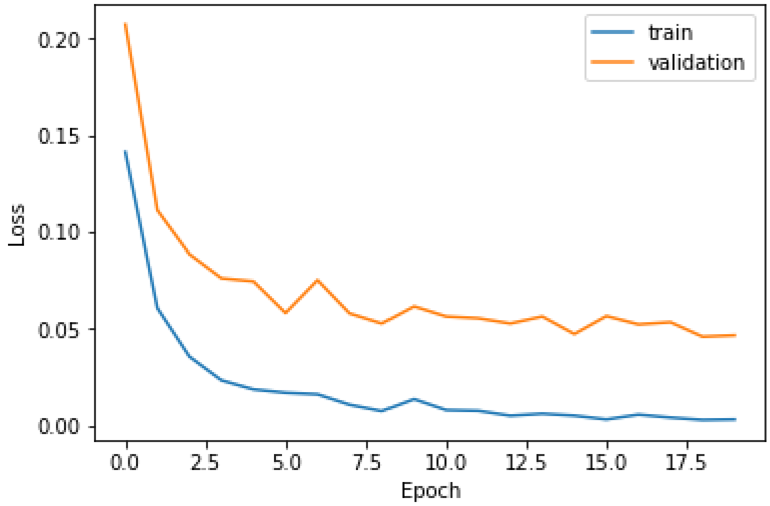
```
WARNING:tensorflow:From <ipython-input-7-f2f5b73a22b4>:18: softmax_cro
ss_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecat
ed and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Epoch  0: train accuracy: 0.859, validation accuracy: 0.793
Epoch  1: train accuracy: 0.939, validation accuracy: 0.889
Epoch  2: train accuracy: 0.964, validation accuracy: 0.911
Epoch  3: train accuracy: 0.977, validation accuracy: 0.924
Epoch  4: train accuracy: 0.981, validation accuracy: 0.925
Epoch  5: train accuracy: 0.983, validation accuracy: 0.942
Epoch  6: train accuracy: 0.984, validation accuracy: 0.925
Epoch  7: train accuracy: 0.989, validation accuracy: 0.942
Epoch  8: train accuracy: 0.992, validation accuracy: 0.947
Epoch  9: train accuracy: 0.986, validation accuracy: 0.938
Epoch 10: train accuracy: 0.992, validation accuracy: 0.944
Epoch 11: train accuracy: 0.992, validation accuracy: 0.944
Epoch 12: train accuracy: 0.995, validation accuracy: 0.947
Epoch 13: train accuracy: 0.994, validation accuracy: 0.944
Epoch 14: train accuracy: 0.995, validation accuracy: 0.953
Epoch 15: train accuracy: 0.997, validation accuracy: 0.943
Epoch 16: train accuracy: 0.994, validation accuracy: 0.948
Epoch 17: train accuracy: 0.996, validation accuracy: 0.946
Epoch 18: train accuracy: 0.997, validation accuracy: 0.954
```

```
Epoch 19: train accuracy: 0.997, validation accuracy: 0.953
Model saved.
```




```
INFO:tensorflow:Restoring parameters from ../../data/traffic-signs/tra
ffic_signs_net
Test accuracy: 0.940
```

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.

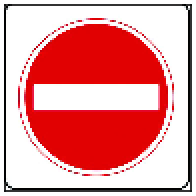
You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

Load and Output the Images

In [8]:
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
import glob
import cv2
import os

image_files = glob.glob( "data/*")
image_files.sort()
rows = len(image_files)
fig, axes = plt.subplots(rows, 1, figsize=(2, 2*rows), squeeze=False)
images = []
for i, image_file in enumerate(image_files):
  img = cv2.imread(image_file)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  axes[i, 0].imshow(img)
  img_name = os.path.basename(image_file)
  images.append([img, img_name])
  axes[i, 0].set_title(img_name)
  axes[i, 0].set_xticks([])
  axes[i, 0].set_yticks([])
```

no-entry.jpg



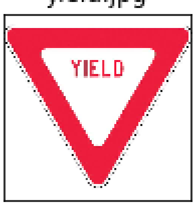
pedestrians.jpg

speed-limit-30.jpeg

stop_sign.jpg

warning.png

yield.jpg



Predict the Sign Type for Each Image

In [9]:
```
### Run the predictions here and use the model to output the prediction
### Make sure to pre-process the images with the same pre-processing pi
### Feel free to use as many code cells as needed.
def predict_images(vars, images):
  saver = tf.train.Saver()
  logits = []
  predictions = []
  with tf.Session() as sess:
    saver.restore(sess, DATA_DIR + "traffic_signs_net")
    rows = len(images)
    fig, axes = plt.subplots(rows, 1, figsize=(2, 2*rows), squeeze=Fals
    for i, (raw_img, name) in enumerate(images):
      img = cv2.resize(raw_img, (32, 32))
      img = normalize(img)
      logit, prediction = sess.run([vars["logits"], vars["prediction_op
                                   feed_dict={vars["X"]: img.reshape(1
      logits.append(logit[0])
      predictions.append(prediction[0])
      axes[i, 0].imshow(raw_img)
      title = "%s: class=%d, %s" % (name, prediction, signnames[predict
      axes[i, 0].set_title(title)
      axes[i, 0].set_xticks([])
      axes[i, 0].set_yticks([])
  return np.array(logits), np.array(predictions)

logits, predictions = predict_images(vars, images)
```

```
INFO:tensorflow:Restoring parameters from ../../data/traffic-signs/tra
ffic_signs_net
```

no-entry.jpg: class=17, No entry

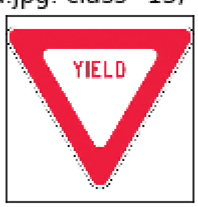
pedestrians.jpg: class=18, General caution

speed-limit-30.jpeg: class=1, Speed limit (30km/h)

stop_sign.jpg: class=14, Stop

warning.png: class=18, General caution

yield.jpg: class=13, Yield



Analyze Performance

In [10]:

```
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it'
true_labels=np.array([17, 27, 1, 14, 18, 13])

predictions = predictions
accuracy = np.mean(predictions==true_labels)
print("Accuracy: %.2f" % accuracy)
```

```
Accuracy: 0.83
```

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.

`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.

In [11]:

```
### Print out the top five softmax probabilities for the predictions on
### Feel free to use as many code cells as needed.
pd.set_option("display.width", 1000)
pd.options.display.float_format = '{:,.3g}'.format
with tf.Session() as sess:
  probs = tf.nn.softmax(tf.constant(logits))
  topk_probs, topk_classes = sess.run(tf.nn.top_k(probs, k=5))
  names = [os.path.basename(f) for f in image_files]
  df = pd.DataFrame({"name": names})
  for i in range(5):
    df["prob_%d" % i] = topk_probs[:,i]
  df["class_0"] = [signnames[id] for id in topk_classes[:,0]]
  df["class_1"] = [signnames[id] for id in topk_classes[:,1]]
  print(df)
```

```
                  name  prob_0   prob_1   prob_2   prob_3   prob_4  
class_0             class_1
0         no-entry.jpg       1 2.49e-16 2.42e-17 8.68e-19 6.78e-22  
No entry            Bumpy road
1      pedestrians.jpg   0.996  0.00408 7.39e-08 4.71e-08 5.28e-09  
General caution         Pedestrians
2   speed-limit-30.jpeg   0.945   0.0552 0.000227 5.27e-08  1.1e-08  Sp
eed limit (30km/h)  Speed limit (80km/h)
3        stop_sign.jpg       1 0.000401 2.76e-05  5.3e-06 1.01e-06  
Stop  Speed limit (80km/h)
4          warning.png       1 4.94e-07 1.38e-07 7.19e-08 1.37e-09  
General caution         Pedestrians
5            yield.jpg       1 2.24e-23 6.85e-26 2.25e-27 7.29e-28  
Yield  Speed limit (30km/h)
```

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

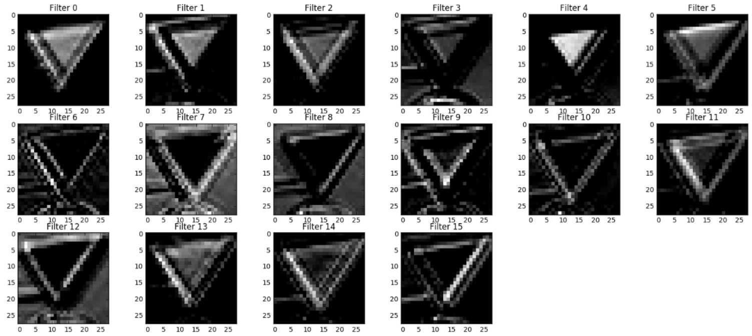
successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

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.



Your output should look something like this (above)

In [12]:
```
### 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
# tf_activation: should be a tf variable name used during your training
# activation_min/max: can be used to view the activation contrast in mo
# plt_num: used to plot out multiple different weight feature map sets 

def outputFeatureMap(image_input, tf_activation, activation_min=-1, act
    # Here make sure to preprocess your image_input in a way your netwo
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow data
    # If you get an error tf_activation is not defined it may be having
    activation = tf_activation.eval(session=sess,feed_dict={x : image_in
    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 map
        plt.title('FeatureMap ' + str(featuremap)) # displays the featu
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nea
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nea
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nea
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nea
```