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 [26]:
```
# Load pickled data
import pickle

# 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) 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 [27]:
```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results
import numpy as np

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

# TODO: Number of validation examples
n_validation = X_valid.shape[0]

# TODO: Number of testing examples.
n_test = X_test.shape[0]

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

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

print("Number of training examples =", n_train)
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 testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

In [155]:
```
n_validation
```

Out[155]:
```
4410
```

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.

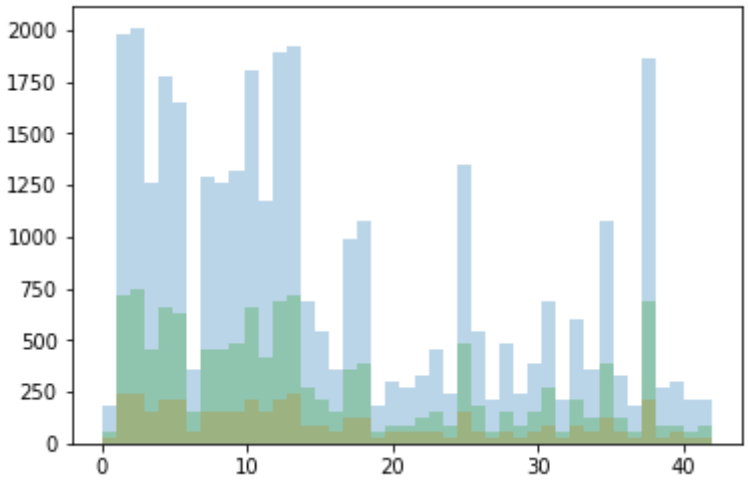
The Matplotlib (http://matplotlib.org/) examples (http://matplotlib.org/examples/index.html) and gallery (http://matplotlib.org/gallery.html) pages are a great resource for doing visualizations in Python.

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 [28]:
```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import numpy as np
# Visualizations will be shown in the notebook.
%matplotlib inline
```

In [29]:
```
# Draw some random traffic signs from the training set
import random
fig, axes = plt.subplots(2,5, figsize = (22,8))
fig.tight_layout()
axes = axes.ravel()

#i_label = []
for i in range(len(axes)):
    rand_fig_index = random.randint(0, len(X_train))
    j = y_train[rand_fig_index]
    #if j not in i_label:
    image = X_train[rand_fig_index]
    axes[i].imshow(image)
    axes[i].set_title(j)
    #i_label.append(j)
```

In [30]:
```
# Plot different classes as histogram
kwargs = dict(alpha=0.3, bins=n_classes)
plt.hist(y_train, **kwargs)
plt.hist(y_valid, **kwargs)
plt.hist(y_test, **kwargs)
plt.show()
```



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.)

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.

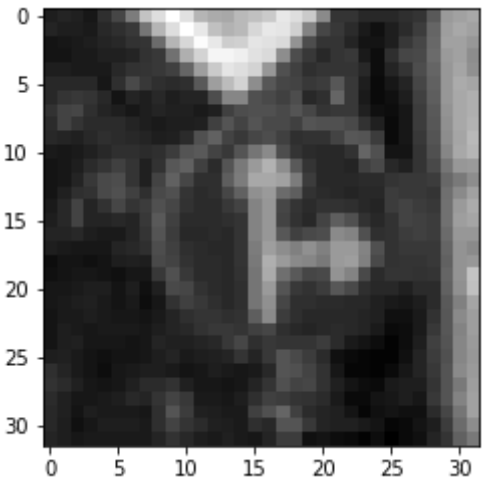
Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

In [31]:
```
### Preprocess the data here. It is required to normalize the data. Other pr
### converting to grayscale, etc.
### Feel free to use as many code cells as needed.
```

In [32]:
```
# RGB images into grayscale
import tensorflow as tf
def rgbtogray(images):
    with tf.Session() as sess:
        images_coverted = sess.run(tf.image.rgb_to_grayscale(images))
    return images_coverted
```

In [33]:
```
# Testing the function rgbtogray
img = rgbtogray(X_train[1000])
print(img.shape)
plt.imshow(np.squeeze(img), cmap='gray')
plt.show()
```

```
(32, 32, 1)
```



In [34]:
```
X_train = rgbtogray(X_train)
X_valid = rgbtogray(X_valid)
X_test = rgbtogray(X_test)
print ('rbg to grayscale done!')
```

```
rbg to grayscale done!
```

In [35]:
```
# Normalize the data
X_train = X_train - np.mean(X_train)
X_valid = X_valid - np.mean(X_valid)
X_test = X_test - np.mean(X_test)
print ('Normalization done!')
```

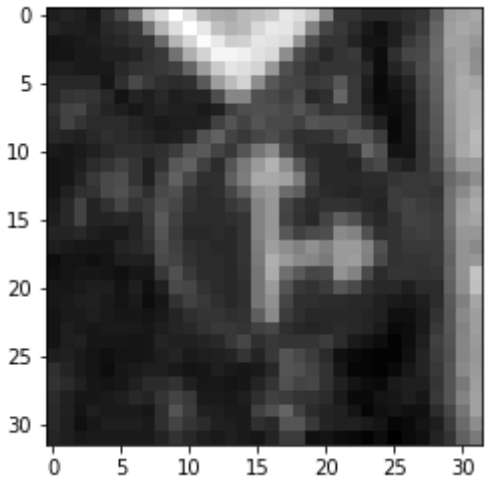
```
Normalization done!
```

In [11]:
```
# Image augmentation, getting more training image data
```

In [37]:
```
# Brightness adjustment method
def img_bright(img):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        img_randbright = sess.run(tf.image.random_brightness(img, max_delta
    return img_randbright
```

In [38]:
```
img = img_bright(X_train[1000])
plt.imshow(np.squeeze(img), cmap='gray')
```

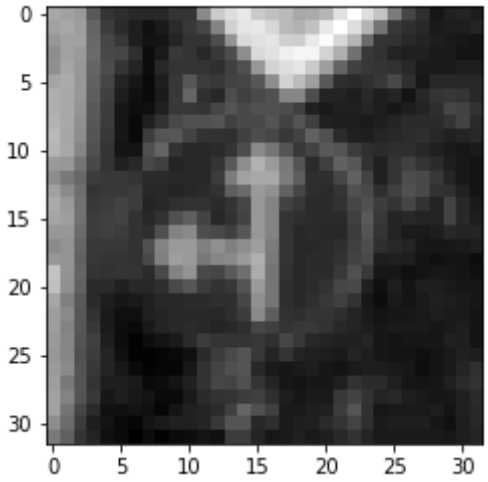
Out[38]:
```
<matplotlib.image.AxesImage at 0x1c3f646940>
```



In [39]:
```
# Hrizontal flipping images
def img_hflip(img):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        img_hflip = sess.run(tf.image.flip_left_right(img))
    return img_hflip
```

In [40]:
```
img = img_hflip(X_train[1000])
plt.imshow(np.squeeze(img), cmap='gray')
```

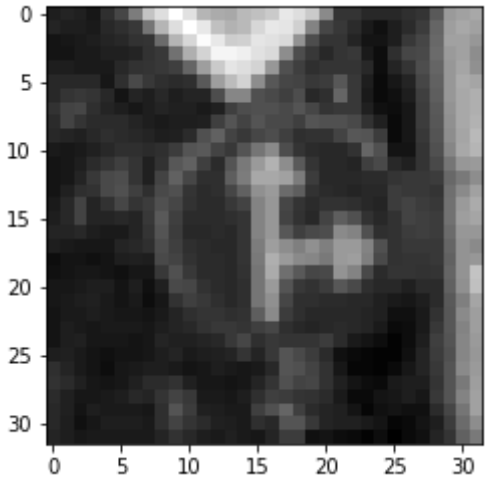
Out[40]: <matplotlib.image.AxesImage at 0x1c3f74ffd0>



In [41]:
```
# Contrat adjust images
def img_contrast(img):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        img_contrasted = sess.run(tf.image.random_contrast(img, lower=1, upp
    return img_contrasted
```

In [42]:
```
img = img_contrast(X_train[1000])
plt.imshow(np.squeeze(img), cmap='gray')
```

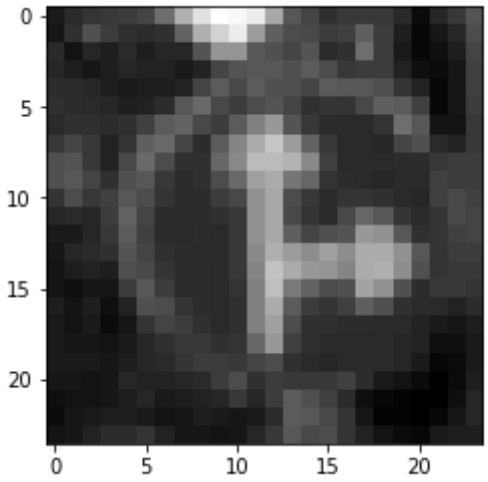
Out[42]: <matplotlib.image.AxesImage at 0x1c3f53fdd8>



In [43]:
```
# Central crop images
def img_ccrop(img):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        img_croped = sess.run(tf.image.central_crop(img, central_fraction=0.
    return img_croped
```

In [44]:
```
img = img_ccrop(X_train[1000])
plt.imshow(np.squeeze(img), cmap='gray')
```

Out[44]:
```
<matplotlib.image.AxesImage at 0x1c2aac6470>
```



In [45]:
```
X_train.shape, y_train.shape
```

Out[45]:
```
((34799, 32, 32, 1), (34799,))
```

In [46]:
```
X_train[1000].shape
```

Out[46]:
```
(32, 32, 1)
```

In [47]:
```
# Augment the train images
'''
The training data needed is 10 x 128. The average number of median sized tra
And the upper number of validation classes is 750.
While the lower number of the training classes is 250.
In consideration of the sizes above, augment the lower parts of the training
'''
'''
The brightness and contrast adjustment will be applied to all the operations
the other one with central cropping. All these will add up to about 750 imag
'''
'''
def DataAug_flip(images):
    images_bright = img_bright(images)
    images_flip = img_hflip(images_bright)
    images_flip = img_contrast(images_flip)
    return images_flip


def DataAug_crop(images):
    images_bright = img_bright(images)
    images_crop = img_ccrop(images_bright)
    images_crop = img_contrast(images_crop)
    return images_crop

X_train_aug = []
y_train_aug = []

for class_n in range(1):
    pictures = np.where(y_train == class_n)
    samples_num = len(pictures[0])

    if samples_num < 500:
        for images in range(samples_num):
            images_flipped = DataAug_flip(X_train[images])
            #images_cropped = DataAug_crop(images)
            X_train_aug.append(images_flipped)
            #X_train_aug.append(images_cropped)
            y_train_aug.append(class_n)
            #y_train_aug.append(class_n)
return X_train_aug, y_train_aug


print ('Data augment done!')
'''
```

Out[47]:
```
"\ndef DataAug_flip(images):\n    images_bright = img_bright(images)\n
images_flip = img_hflip(images_bright)\n    images_flip = img_contrast(im
ages_flip)\n    return images_flip\n\n\ndef DataAug_crop(images):\n    im
ages_bright = img_bright(images)\n    images_crop = img_ccrop(images_brig
ht)\n    images_crop = img_contrast(images_crop)\n    return images_crop
\n\nX_train_aug = []\ny_train_aug = []\n\nfor class_n in range(1):\n    p
ictures = np.where(y_train == class_n)\n    samples_num = len(pictures
[0])\n    \n    if samples_num < 500:\n        for images in range(sample
s_num):\n            images_flipped = DataAug_flip(X_train[images])\n
        #images_cropped = DataAug_crop(images)\n            X_train_aug.a
ppend(images_flipped)\n            #X_train_aug.append(images_cropped)\n
```

```
            y_train_aug.append(class_n)\n                #y_train_aug.append(c
lass_n)\nreturn X_train_aug, y_train_aug\n\n                \nprint ('Data au
gment done!')\n"
```

In [25]:
```
# Data augment, two ways of data aument, take too much time to be done, used
'''
    if samples_num <= 400:
        for i in range(samples_num):
            img_num = pictures[0][i]
            image = img_bright(X_train[img_num])
            X_train_aug.append(image)
            y_train_aug.append(class_n)
'''

X_train_aug = []
y_train_aug = []

for class_n in range(n_classes):
    pictures = np.where(y_train == class_n)
    samples_num = len(pictures[0])

    if samples_num <= 300:
        for i in range(samples_num):
            img_num = pictures[0][i]
            image = img_hflip(X_train[img_num])
            X_train_aug.append(image)
            y_train_aug.append(class_n)
    else:
        print ('Class ', class_n, ' is not augmented!')
#return X_train_aug, y_train_aug
print ('Partial Augment done!')
```

```
Class  1  is not augmented!
Class  2  is not augmented!
Class  3  is not augmented!
Class  4  is not augmented!
Class  5  is not augmented!
Class  6  is not augmented!
Class  7  is not augmented!
Class  8  is not augmented!
Class  9  is not augmented!
Class  10  is not augmented!
Class  11  is not augmented!
Class  12  is not augmented!
Class  13  is not augmented!
Class  14  is not augmented!
Class  15  is not augmented!
Class  16  is not augmented!
Class  17  is not augmented!
Class  18  is not augmented!
Class  22  is not augmented!
Class  23  is not augmented!
Class  25  is not augmented!
Class  26  is not augmented!
Class  28  is not augmented!
Class  30  is not augmented!
Class  31  is not augmented!
Class  33  is not augmented!
Class  34  is not augmented!
Class  35  is not augmented!
Class  36  is not augmented!
```

```
Class  38  is not augmented!
Partial Augment done!
```

In [26]:
```
np.array(X_train_aug).shape
```

Out[26]:
```
(3000, 32, 32, 1)
```

In [27]:
```
X_train = np.append(X_train, X_train_aug, axis=0)
y_train = np.append(y_train, y_train_aug, axis=0)
```

In [28]:
```
X_train.shape
```

Out[28]:
```
(37799, 32, 32, 1)
```

Model Architecture

In [29]:
```
### Define your architecture here.
### Feel free to use as many code cells as needed.
```

In [48]:
```
# Shuffle the training data set
from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train)
```

In [49]:
```
# Import tensorflow and set the hyperparamters
import tensorflow as tf

EPOCHS = 15
BATCH_SIZE = 128
keep_prob = tf.placeholder(tf.float32)
```

In [50]:
```
# Set the archetiete of TrafficNet
from tensorflow.contrib.layers import flatten

def TrafficNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for
    mu = 0
    sigma = 0.1

    # TODO: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_w = tf.Variable(tf.truncated_normal((5,5,1,6), mean = mu, stddev =
    conv1_b =tf.Variable(tf.zeros(6))
    conv1 = tf.nn.conv2d(x, conv1_w, strides=[1,1,1,1], padding='VALID') + c

    # TODO: Activation.
    conv1 = tf.nn.relu(conv1)

    # TODO: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=(1,2,2,1), strides=(1,2,2,1), paddir
    conv1 = tf.nn.dropout(conv1, keep_prob)

    # TODO: Layer 2: Convolutional. Output = 10x10x16.
    conv2_w = tf.Variable(tf.truncated_normal((5,5,6,16), mean = mu, stddev
    conv2_b = tf.Variable(tf.zeros(16))
    conv2 = tf.nn.conv2d(conv1, conv2_w, strides=[1,1,1,1], padding = 'VALI

    # TODO: Activation.
    conv2 = tf.nn.relu(conv2)


    # TODO: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=(1,2,2,1), strides=(1,2,2,1), paddir
    conv2 = tf.nn.dropout(conv2, keep_prob)

    # TODO: Flatten. Input = 5x5x16. Output = 400.
    conv2 = flatten(conv2)

    # TODO: Layer 3: Fully Connected. Input = 400. Output = 240.
    fc1_w = tf.Variable(tf.truncated_normal((400,240), mean=mu, stddev=sigma
    fc1_b = tf.Variable(tf.zeros(240))
    fc1 = tf.matmul(conv2, fc1_w) + fc1_b
    # TODO: Activation.
    fc1 = tf.nn.relu(fc1)
    fc1 = tf.nn.dropout(fc1, keep_prob)


    # TODO: Layer 4: Fully Connected. Input = 240. Output = 160.
    fc2_w = tf.Variable(tf.truncated_normal((240,160), mean=mu, stddev=sigma
    fc2_b = tf.Variable(tf.zeros(160))
    fc2 = tf.matmul(fc1, fc2_w) + fc2_b
    # TODO: Activation.
    fc2 = tf.nn.relu(fc2)
    fc2 = tf.nn.dropout(fc2, keep_prob)

    # TODO: Layer 5: Fully Connected. Input = 160. Output = 43.
    fc3_w = tf.Variable(tf.truncated_normal((160,43), mean=mu, stddev=sigma)
    fc3_b = tf.Variable(tf.zeros(43))
```

```
    logits = tf.matmul(fc2, fc3_w) + fc3_b

    return logits
```

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 [51]:
```
### 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.
```

In [52]:
```
x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
```

In [53]:
```
rate = 0.001

logits = TrafficNet(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, lo
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)
```

In [54]:
```
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))
saver = tf.train.Saver()

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:o
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: ba
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

In [55]:
```
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)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

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

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.385

EPOCH 2 ...
Validation Accuracy = 0.758

EPOCH 3 ...
Validation Accuracy = 0.856

EPOCH 4 ...
Validation Accuracy = 0.895

EPOCH 5 ...
Validation Accuracy = 0.920

EPOCH 6 ...
Validation Accuracy = 0.915

EPOCH 7 ...
Validation Accuracy = 0.935

EPOCH 8 ...
Validation Accuracy = 0.940

EPOCH 9 ...
Validation Accuracy = 0.941

EPOCH 10 ...
Validation Accuracy = 0.942

EPOCH 11 ...
Validation Accuracy = 0.939

EPOCH 12 ...
```

```
Validation Accuracy = 0.948

EPOCH 13 ...
Validation Accuracy = 0.952

EPOCH 14 ...
Validation Accuracy = 0.951

EPOCH 15 ...
Validation Accuracy = 0.952

Model saved
```

In [56]:
```
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))
```

```
INFO:tensorflow:Restoring parameters from ./trafficnet
Test Accuracy = 0.926
```

In [156]:
```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

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

```
INFO:tensorflow:Restoring parameters from ./trafficnet
Test Accuracy = 0.994
```

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.

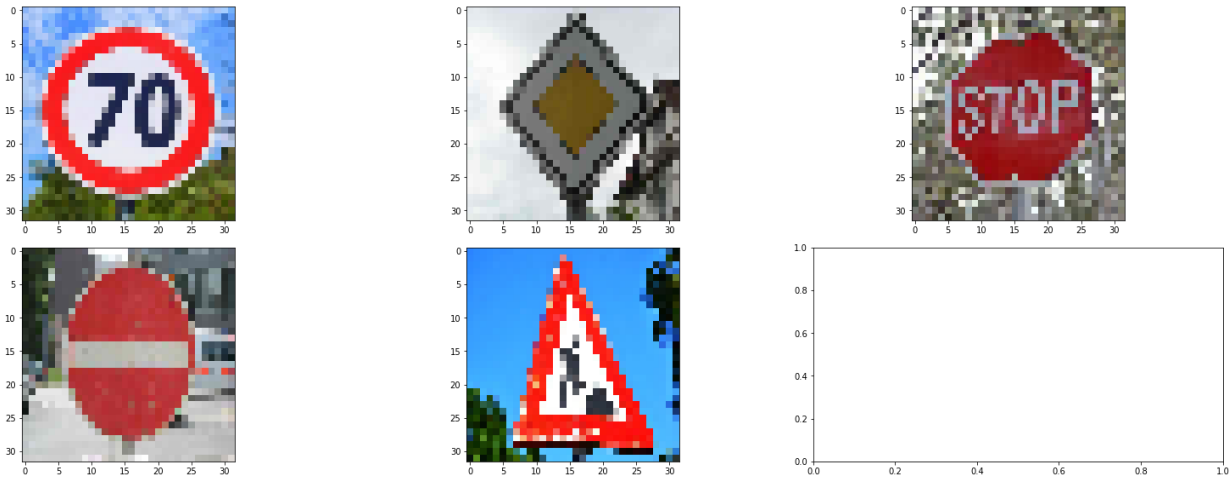
You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

Load and Output the Images

In [171]:
```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
```

In [157]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
# Visualizations will be shown in the notebook.
%matplotlib inline

# Layout the subplots
fig, axes = plt.subplots(2,3, figsize=(22,8))
fig.tight_layout()
axes = axes.ravel()

# Read the test images
img0 = mpimg.imread('test-data/4.jpg')
img1 = mpimg.imread('test-data/12.jpg')
img2 = mpimg.imread('test-data/14.jpg')
img3 = mpimg.imread('test-data/17.jpg')
img4 = mpimg.imread('test-data/25.jpg')

img0 = cv2.resize(img0, (32,32))
img1 = cv2.resize(img1, (32,32))
img2 = cv2.resize(img2, (32,32))
img3 = cv2.resize(img3, (32,32))
img4 = cv2.resize(img4, (32,32))

# Display the test images in the subplots
axes[0].imshow(img0)
axes[1].imshow(img1)
axes[2].imshow(img2)
axes[3].imshow(img3)
axes[4].imshow(img4)
```

Out[157]: <matplotlib.image.AxesImage at 0x1c328358d0>

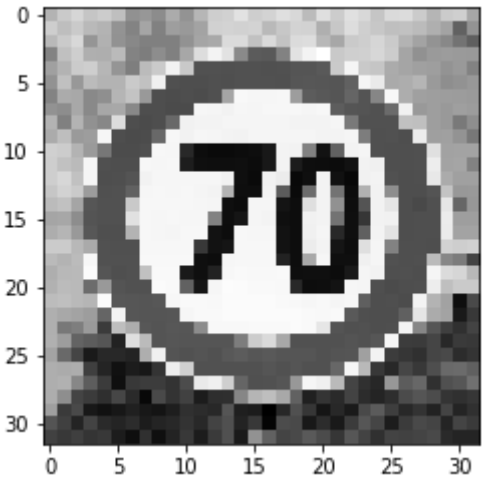


In [158]:
```
# Collect the test data set
test_data = np.array([img0, img1, img2, img3, img4])
test_data.shape
```

Out[158]: (5, 32, 32, 3)

In [159]:
```
# Preprocessing the test images
import numpy as np
test_data_p = rgbtogray(test_data)
test_data_p = test_data_p - np.mean(test_data_p)
print(test_data_p.shape)

plt.imshow(np.squeeze(test_data_p[0]), cmap='gray')
plt.show()
```

```
(5, 32, 32, 1)
```



Predict the Sign Type for Each Image

In [160]:
```
### 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.
```

In [161]:
```
# Read the sign number and name list
import pandas as pd

TrafficSignNames = pd.read_csv('signnames.csv')
TrafficSignNames
```

Out[161]:

	ClassId	SignName
0	0	Speed limit (20km/h)
1	1	Speed limit (30km/h)
2	2	Speed limit (50km/h)
3	3	Speed limit (60km/h)
4	4	Speed limit (70km/h)
5	5	Speed limit (80km/h)
6	6	End of speed limit (80km/h)
7	7	Speed limit (100km/h)
8	8	Speed limit (120km/h)
9	9	No passing
10	10	No passing for vehicles over 3.5 metric tons
11	11	Right-of-way at the next intersection
12	12	Priority road
13	13	Yield
14	14	Stop
15	15	No vehicles
16	16	Vehicles over 3.5 metric tons prohibited
17	17	No entry
18	18	General caution
19	19	Dangerous curve to the left
20	20	Dangerous curve to the right
21	21	Double curve
22	22	Bumpy road
23	23	Slippery road
24	24	Road narrows on the right
25	25	Road work
26	26	Traffic signals
27	27	Pedestrians
28	28	Children crossing
29	29	Bicycles crossing

	ClassId	SignName
30	30	Beware of ice/snow
31	31	Wild animals crossing
32	32	End of all speed and passing limits
33	33	Turn right ahead
34	34	Turn left ahead
35	35	Ahead only
36	36	Go straight or right
37	37	Go straight or left
38	38	Keep right
39	39	Keep left
40	40	Roundabout mandatory
41	41	End of no passing
42	42	End of no passing by vehicles over 3.5 metric ...

In [162]:
```
test_data_label = [4, 12, 14, 17, 25]
```

Analyze Performance

In [163]:
```
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20
```

In [164]:
```
import tensorflow as tf
prob = tf.nn.softmax(logits)

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('./'))
    predicted_class = sess.run(prob, feed_dict={x: test_data_p, keep_prob: 1
```

```
INFO:tensorflow:Restoring parameters from ./trafficnet
```

In [165]:
```
predicted_class.shape
```

Out[165]:
```
(5, 43)
```

In [166]:
```
with tf.Session() as sess:
    values, indices = sess.run(tf.nn.top_k(predicted_class, k=5))
```

In [170]:
```
values
```

Out[170]:
```
array([[9.9919492e-01, 8.0514408e-04, 2.2517412e-08, 1.9311186e-08,
        1.7599916e-11],
       [1.0000000e+00, 7.8176032e-10, 1.2372714e-10, 5.5255435e-14,
        2.3216055e-14],
       [9.9991286e-01, 7.4698124e-05, 1.1673747e-05, 6.2291610e-07,
        2.1093950e-08],
       [1.0000000e+00, 7.2525583e-14, 5.6742075e-16, 7.9534627e-17,
        5.6615286e-19],
       [1.0000000e+00, 5.8627725e-09, 4.3851359e-10, 6.2914587e-12,
        3.2088104e-13]], dtype=float32)
```

In [167]:
```
indices
```

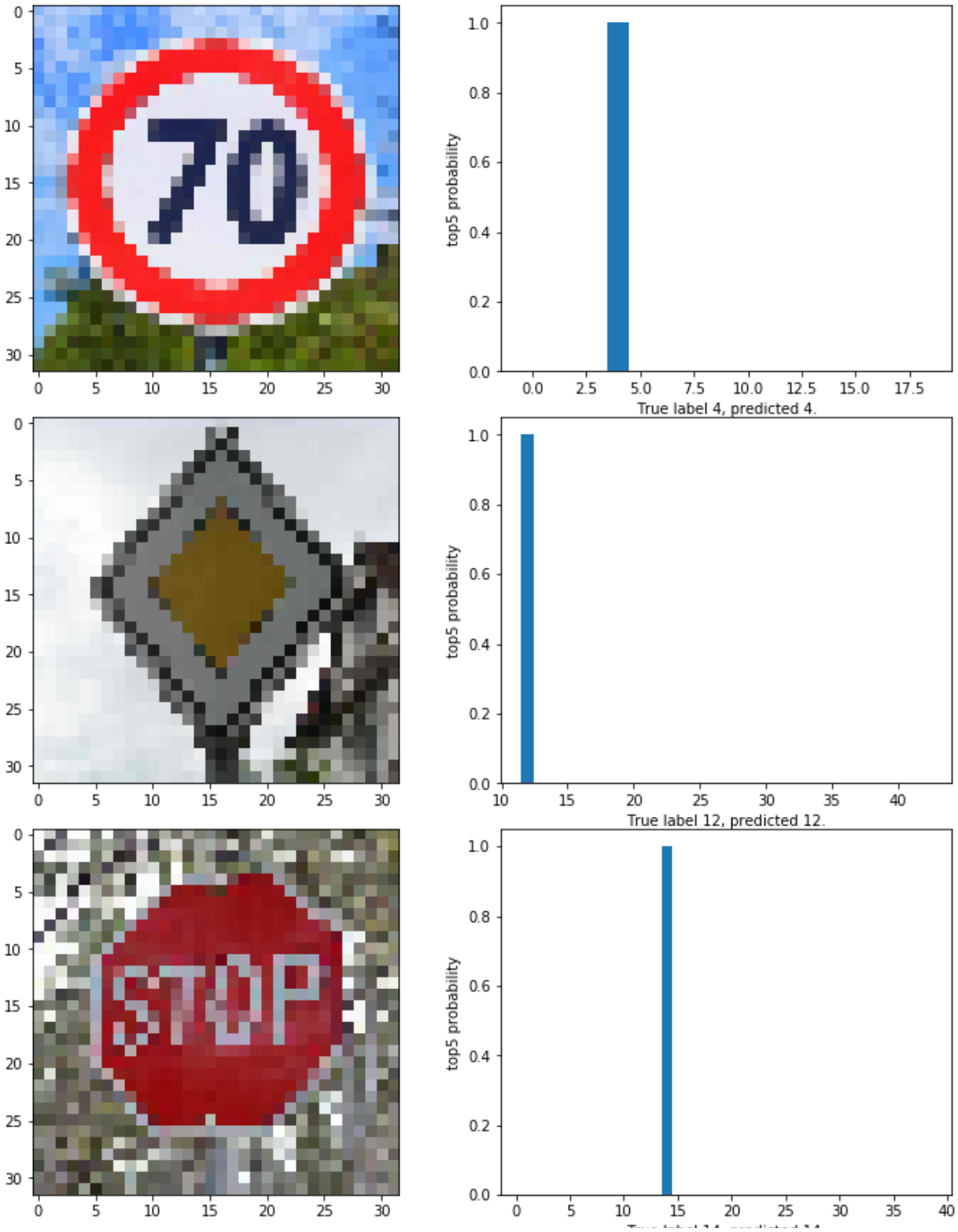
Out[167]:
```
array([[ 4,  0,  6, 18,  5],
       [12, 42, 32, 13, 40],
       [14, 38, 34,  1, 17],
       [17, 34,  9, 14, 13],
       [25, 30, 24, 29, 22]], dtype=int32)
```

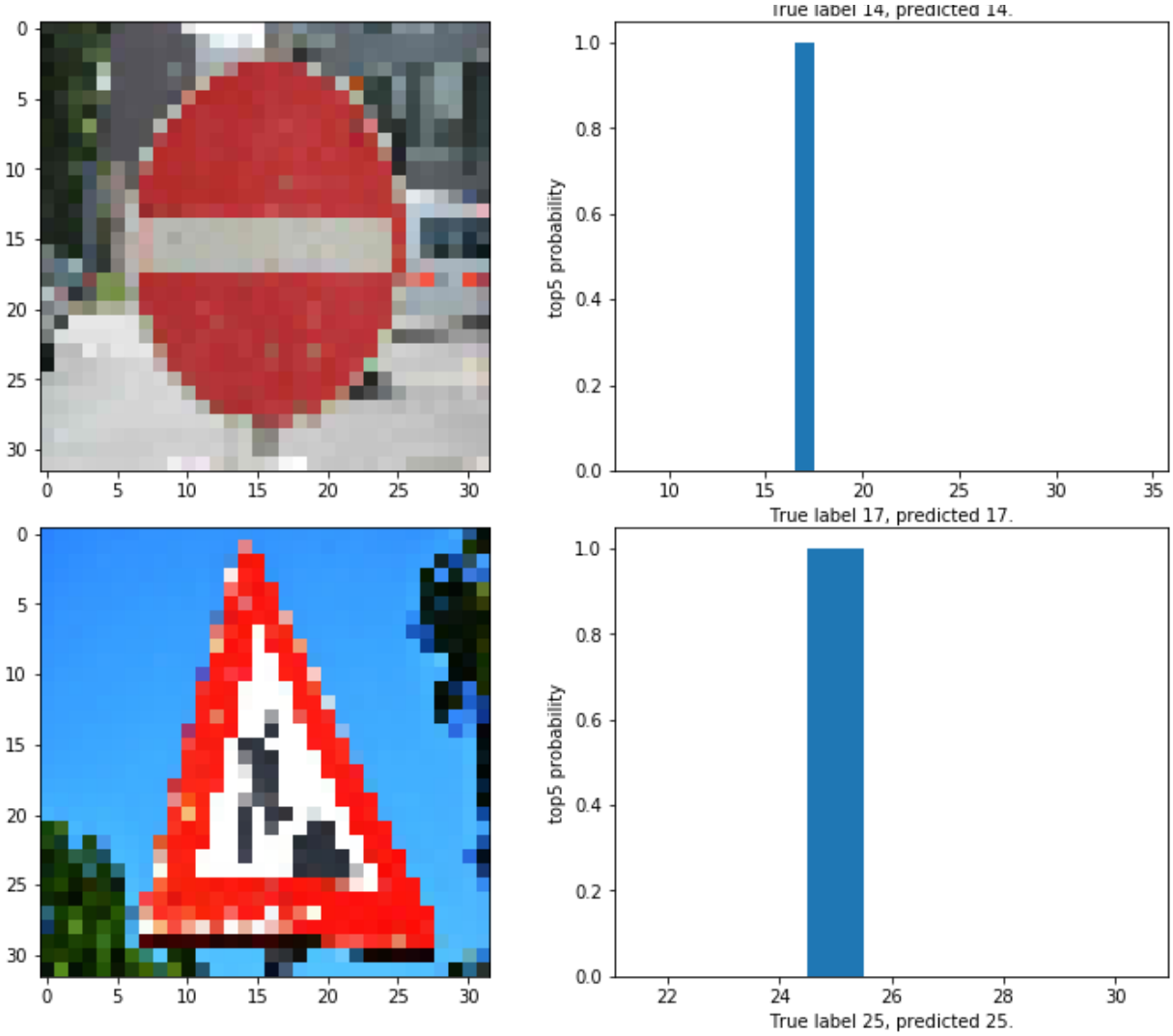
In [168]:
```
# The accuracy of classification of the 5 test images
with tf.Session() as sess:
    accuracy = sess.run(tf.equal(test_data_label, indices[:,0]))
print (accuracy)
accuracy = sum(accuracy)/len(accuracy)
accuracy
```

```
[ True  True  True  True  True]
```

Out[168]:
```
1.0
```

In [172]:
```
# Plot images along side their predictions
fig, axes = plt.subplots(5,2, figsize=(10, 20))
fig.tight_layout()
for i in range(len(predicted_class)):

    axes[i,0].imshow(test_data[i])
    axes[i,1].bar(indices[i], height = values[i], width = 1.0)
    axes[i,1].set_ylabel('top5 probability')
    axes[i,1].set_xlabel('True label {tlabel}, predicted {plabel}.' .format(
```





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

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)](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.