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. Sections that begin with **'Implementation'** in the header indicate where you should begin your implementation for your project. Note that some sections of implementation are optional, and will be marked with **'Optional'** in the header.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **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 [18]:
```
# Load pickled data
import pickle

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

training_file = '/Users/mckain/Projects/CarND-Traffic-Sign-Classifier-Projec
testing_file = '/Users/mckain/Projects/CarND-Traffic-Sign-Classifier-Project

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

X_train, y_train = train['features'], train['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 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.

In [19]:
```
### Replace each question mark with the appropriate value.

# TODO: Number of training examples
n_train = len(X_train)

# TODO: Number of testing examples.
n_test = len(X_test)

# TODO: What's the shape of an traffic sign image?
image_shape = X_train[0].shape

# TODO: How many unique classes/labels there are in the dataset.
n_classes = 1 + y_train.max()

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 = 39209
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

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 [20]:
```
### Data exploration visualization goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import random
# Visualizations will be shown in the notebook.
%matplotlib inline

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

plt.figure(figsize=(1,1))
plt.imshow(image)
print('label id: ', y_train[index])
```

```
label id:  13
```

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).

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

- Neural network architecture
- 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.

NOTE: 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!

Implementation

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project. Once you have completed your implementation and are satisfied with the results, be sure to thoroughly answer the questions that follow.

In [21]:
```
# ########################################################
### Preprocess the data here.
### Feel free to use as many code cells as needed.
# ########################################################

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_t

X_train, y_train = shuffle(X_train, y_train)  # Suffle it

# ########################################################
# Set up TensorFlow
# ########################################################
import tensorflow as tf

EPOCHS = 10
BATCH_SIZE = 128

# ########################################################
# Using LeNet in my pipeline
# ########################################################
from tensorflow.contrib.layers import flatten

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

    # Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, 6), mean = mu,
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID'

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

    # Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],

    # Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VA

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

    # Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],

    # Flatten. Input = 5x5x16. Output = 400.
    fc0   = flatten(conv2)
```

```
    # Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stc
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

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

    # Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stc
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

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

    # Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = mu, stdc
    fc3_b  = tf.Variable(tf.zeros(43))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

In [22]:
```
# Features & Labels
x = tf.placeholder(tf.float32, (None, 32, 32, 3))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
```

Training Pipeline

In [23]:
```
rate = 0.001

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

Model Evaluation method

In [24]:
```
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
```

Training the model

In [25]:
```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    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_validation, y_validation)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

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

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.720

EPOCH 2 ...
Validation Accuracy = 0.839

EPOCH 3 ...
Validation Accuracy = 0.891

EPOCH 4 ...
Validation Accuracy = 0.913

EPOCH 5 ...
Validation Accuracy = 0.926

EPOCH 6 ...
Validation Accuracy = 0.939

EPOCH 7 ...
Validation Accuracy = 0.935

EPOCH 8 ...
Validation Accuracy = 0.944

EPOCH 9 ...
Validation Accuracy = 0.946

EPOCH 10 ...
Validation Accuracy = 0.948

Model saved
```

Question 1

Describe how you preprocessed the data. Why did you choose that technique?

Answer:

I used the below steps to preprocess the data:

- **Split the data into training/validation/testing**
  - I used 20% of the training set and splitted into a new set called 'validation'.
- **Shuffle:** I used `shuffle` to mix the data around, to help training the model.

Question 2

*Describe how you set up the training, validation and testing data for your model. **Optional**: If you generated additional data, how did you generate the data? Why did you generate the data? What are the differences in the new dataset (with generated data) from the original dataset?*

Answer:

Architecture

I used the powerful LeNet architecture based on the same LeNet Lab from the class. However I had to change a few things to make this work.

- The source of the training & testing set it's different, I had to download these 2 files to my local computer and load it from there.
- The training/testing set comes in colors, so the channel was switched to **3**
- The labels or unique classes are different from the lab (of course).
- There was no validation set, so I extracted 20% of the training set as the validation one.

Question 3

What does your final architecture look like? (Type of model, layers, sizes, connectivity, etc.) For reference on how to build a deep neural network using TensorFlow, see Deep Neural Network in TensorFlow (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/b516a270-8600-4f93-a0a3-20dfeabe5da6/concepts/83a3a2a2-a9bd-4b7b-95b0-eb924ab14432) from the classroom.

Answer:

My architecture looks like:

- 2 Convolutional layers,
  - 1: Input = 32x32x1. Output = 28x28x6.
  - 2: Output= 10x10x16

- 2 Pooling layers, rigt after each Convolutional layer to help to reduce the spatial size
  - 1: Output = 14x14x6
  - 2: Output = 5x5x6
- 3 Fully Connected layers.
  - 1: Input = 400. Output = 120.
  - 2: Input = 120. Output = 84.
  - 3: Input = 84. Output = 10.

Question 4

How did you train your model? (Type of optimizer, batch size, epochs, hyperparameters, etc.)

Answer:

- I used the AdamOptimizer (I heard it's an advanced optimized with better results than the GradientDescendent) with a learning rate of 0.001
- 10 Epochs (It threw a decent accuracy)
- A batch size of 128

Question 5

What approach did you take in coming up with a solution to this problem? It may have been a process of trial and error, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think this is suitable for the current problem.

Answer:

I think the powerful convolutional network LeNet5 it's a suitable architecture to tackle this Traffic Sign Classifier, since it was designed orignally for handwritten and machine-printed character recognition.

What makes a good choice it's the combinations of Layers that transform the image volume into an output volume.

Step 3: Test a Model on New Images

Take several pictures of traffic signs that you find on the web or around you (at least five), and run them through your classifier on your computer to produce example results. The classifier might not recognize some local signs but it could prove interesting nonetheless.

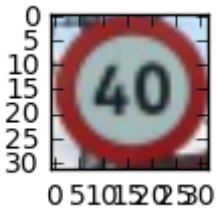
You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

Implementation

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project. Once you have completed your implementation and are satisfied with the results, be sure to thoroughly answer the questions that follow.

In [26]:
```
m PIL import Image
m scipy import ndimage, misc
ort matplotlib.pyplot as plt
isualizations will be shown in the notebook.
tplotlib inline

ge_names = ['./spain-traffic-signs-set/5_degree_sign.jpg', './spain-traffic-s

in_signs = []

 image_name in image_names:
 image = ndimage.imread(image_name, mode="RGB")
 spain_signs.append(image)
 print('Shape: ', image.shape)
 plt.figure(figsize=(1,1))
 plt.imshow(image)

nt(len(spain_signs))
```

```
Shape:  (32, 32, 3)
Shape:  (32, 32, 3)
Shape:  (32, 32, 3)
Shape:  (32, 32, 3)
Shape:  (32, 32, 3)
5
```



Question 6

Choose five candidate images of traffic signs and provide them in the report. Are there any particular qualities of the image(s) that might make classification difficult? It could be helpful to plot the images in the notebook.

Answer:

Well, It seems these traffic signs that I used are from Spain, the images are not the best ones in terms of light conditions and are in different color and shapes than the ones used to train my model.

This is the URL: http://agamenon.tsc.uah.es/Investigacion/gram/traffic_signs.html (http://agamenon.tsc.uah.es/Investigacion/gram/traffic_signs.html)

Question 7

Is your model able to perform equally well on captured pictures when compared to testing on the dataset? The simplest way to do this check the accuracy of the predictions. For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate.

NOTE: *You could check the accuracy manually by using* `signnames.csv` *(same directory). This file has a mapping from the class id (0-42) to the corresponding sign name. So, you could take the class id the model outputs, lookup the name in* `signnames.csv` *and see if it matches the sign from the image.*

Answer:

In [35]:
```
### Visualize the softmax probabilities here.
### Feel free to use as many code cells as needed.
import tensorflow as tf

with tf.Session() as sess:
    loader = tf.train.import_meta_graph('./lenet.meta')
    loader.restore(sess, tf.train.latest_checkpoint('.'))
    spain_predictions = sess.run(tf.argmax(logits, 1), feed_dict={x: spain_s
    print(spain_predictions)

spain_signs_predictions = []
import csv
with open('./signnames.csv', 'r') as csvfile:
    signnamesreader = csv.reader(csvfile)
    for row in signnamesreader:
        spain_signs_predictions.append(row[1])
    spain_signs_predictions = spain_signs_predictions[1:]
    for index, value in enumerate(spain_predictions):
        print('=> ', spain_predictions[index], spain_signs_predictions[value
```

```
[38 23 10  4 11]
=>  38 Keep right
=>  23 Slippery road
=>  10 No passing for vehicles over 3.5 metric tons
=>  4 Speed limit (70km/h)
=>  11 Right-of-way at the next intersection
```

Question 8

*Use the model's softmax probabilities to visualize the **certainty** of its predictions, [tf.nn.top_k (https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k)](https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) could prove helpful here. Which predictions is the model certain of? Uncertain? If the model was incorrect in its initial prediction, does the correct prediction appear in the top k? (k should be 5 at most)*

`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:

```
# (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.

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.

In [ ]: