```
import time
import tensorflow as tf
import numpy as np
import pandas as pd
from scipy.misc import imread
from alexnet import AlexNet

sign_names = pd.read_csv('signnames.csv')
nb_classes = 43

x = tf.placeholder(tf.float32, (None, 32, 32, 3))
resized = tf.image.resize_images(x, (227, 227))

# Returns the second final layer of the AlexNet model,
# this allows us to redo the last layer specifically for
# traffic signs model.
fc7 = AlexNet(resized, feature_extract=True)
shape = (fc7.get_shape().as_list()[-1], nb_classes)
fc8W = tf.Variable(tf.truncated_normal(shape, stddev=1e-2))
fc8b = tf.Variable(tf.zeros(nb_classes))
logits = tf.nn.xw_plus_b(fc7, fc8W, fc8b)
probs = tf.nn.softmax(logits)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

# Read Images
im1 = imread("construction.jpg").astype(np.float32)
im1 = im1 - np.mean(im1)
```

```
im2 = imread("stop.jpg").astype(np.float32)
im2 = im2 - np.mean(im2)

# Run Inference
t = time.time()
output = sess.run(probs, feed_dict={x: [im1, im2]})

# Print Output
```



```
    print("Image", input_im_ind)
    for i in range(5):
        print("%s: %.3f" % (sign_names.ix[inds[-1 - i]][1], output[
input_im_ind, inds[-1 - i]]))
    print()

print("Time: %.3f seconds" % (time.time() - t))
```

Here's how I did it:

```
# Returns the second final layer of the AlexNet model,
# this allows us to redo the last layer specifically for
# traffic signs model.
fc7 = AlexNet(resized, feature_extract=True)
shape = (fc7.get_shape().as_list()[-1], nb_classes)
fc8W = tf.Variable(tf.truncated_normal(shape, stddev=1e-2))
fc8b = tf.Variable(tf.zeros(nb_classes))
logits = tf.nn.xw_plus_b(fc7, fc8W, fc8b)
probs = tf.nn.softmax(logits)
```

First, I figure out the shape of the final fully connected layer, in my opinion this is the trickiest part.

To do that I have to figure out the size of the output from `fc7`. Since it's a fully connected layer I know it's shape will be 2D so the second (or last) element of the list will be the size of the output. `fc7.get_shape().as_list()[-1]` does the trick. I then combine this with the number of classes for the Traffic Sign dataset to get the shape of the final fully connected layer, `shape = (fc7.get_shape().as_list()[-1], nb_classes)`. The rest of the code is just the standard way to define a fully connected in TensorFlow. Finally, I calculate the probabilities via softmax, `probs = tf.nn.softmax(logits)`.

NEXT