Adding in our own transfer layer by hand with AlexNet

In [7]:

```
import random
import tensorflow as tf
import numpy as np
import os
from scipy import ndimage
import matplotlib.pyplot as plt

%matplotlib inline
```

Load in our previous exported model

In [8]:

```
graph = tf.Graph()
with graph.as_default():
    importer = tf.train.import_meta_graph('saved_models/alex_vars.meta')

sess = tf.Session(graph=graph)
importer.restore(sess, 'saved_models/alex_vars')
```

Get handle to second-to-last layer in pre-built model

In [9]:

```
fc7_op = graph.get_operation_by_name('fc7/relu')
fc7 = fc7_op.outputs[0]
```

In [10]:

```
fc7.get_shape()
```

Out[10]:

```
TensorShape([Dimension(None), Dimension(4096)])
```

Create new layer, attached to `fc7`

In [11]:

```
# Create new final layer
with graph.as_default():
    x = graph.get_operation_by_name('input').outputs[0]

    with tf.name_scope('transfer'):
        labels = tf.placeholder(tf.int32, [None])
        one_hot_labels = tf.one_hot(labels, 2)

        with tf.name_scope('cat_dog_final_layer'):
            weights = tf.Variable(tf.truncated_normal([4096, 2],
stddev=0.001),
                                  name='final_weights')
            biases = tf.Variable(tf.zeros([2]), name='final_biases')
            logits = tf.nn.xw_plus_b(fc7, weights, biases, name='logits')

        prediction = tf.nn.softmax(logits, name='cat_dog_softmax')
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits, one_ho
t_labels)
        loss = tf.reduce_mean(cross_entropy, name='cat_dog_loss')

        global_step = tf.Variable(0, trainable=False, name='global_step')
        inc_step = global_step.assign_add(1)

        cat_dog_variables = [weights, biases]
        train = tf.train.GradientDescentOptimizer(0.01).minimize(loss, global_
step=global_step,
                                                                 var_list=cat_d
og_variables)

    with tf.name_scope('accuracy'):
        label_prediction = tf.argmax(prediction, 1, name='predicted_label')
        correct_prediction = tf.equal(label_prediction, tf.argmax(one_hot_labe
ls, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))


    init = tf.initialize_all_variables()
```

In [12]:

```
sess = tf.Session(graph=graph)
sess.run(init)
```

Get our training file names

In [13]:

```
cat_files = [
    'data/dogs_and_cats/cats/' + f
    for
    f
    in
    os.listdir('data/dogs_and_cats/cats')
]

dog_files = [
    'data/dogs_and_cats/dogs/' + f
    for
    f
    in
    os.listdir('data/dogs_and_cats/dogs')
]

all_files = cat_files + dog_files
```

Shuffle and split into training/validation

In [14]:

```
random.shuffle(all_files)
```

In [15]:

```
num_files = len(all_files)
valid_percentage = 0.3
split = int(num_files * valid_percentage)
valid_data = all_files[:split]
train_data = all_files[split:]
```

In [16]:

```
print('Number of training images: {}'.format(len(train_data)))
print('Number of validation images: {}'.format(len(valid_data)))
```

```
Number of training images: 17500
Number of validation images: 7500
```

Create generator to give us batches of data

In [27]:

```
from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat
```

In [28]:

```
flip_left_right = True
random_crop = 1
random_scale = 1
random_brightness = 1
num_channels = 3
height = 227
width = 227
pixel_depth = 255.0
```

In [34]:

```
import ntpath

def get_batch(batch_size, data, max_epochs, should_distort=False):
    distort_graph = tf.Graph()
    with distort_graph.as_default():
        """
        From https://github.com/tensorflow/tensorflow/blob/master/tensorflow/e
xamples/image_retraining/retrain.py
        """
        jpeg_name = tf.placeholder(tf.string, name='DistortJPGInput')
        jpeg_data = tf.read_file(jpeg_name)
        decoded_image = tf.image.decode_jpeg(jpeg_data, channels=3)
        resized_image = tf.image.resize_images(decoded_image, (height, width))
        decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
        decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
        margin_scale = 1.0 + (random_crop / 100.0)
        resize_scale = 1.0 + (random_scale / 100.0)
        margin_scale_value = tf.constant(margin_scale)
        resize_scale_value = tf.random_uniform(tensor_shape.scalar(),
                                             minval=1.0,
                                             maxval=resize_scale)
        scale_value = tf.mul(margin_scale_value, resize_scale_value)
        precrop_width = tf.mul(scale_value, width)
        precrop_height = tf.mul(scale_value, width)
        precrop_shape = tf.pack([precrop_height, precrop_width])
        precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
        precropped_image = tf.image.resize_bilinear(decoded_image_4d,
                                                  precrop_shape_as_int)
        precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[0])
        cropped_image = tf.random_crop(precropped_image_3d,
                                     [width, width,
                                      num_channels])
        if flip_left_right:
            flipped_image = tf.image.random_flip_left_right(cropped_image)
        else:
            flipped_image = cropped_image
        brightness_min = 1.0 - (random_brightness / 100.0)
        brightness_max = 1.0 + (random_brightness / 100.0)
        brightness_value = tf.random_uniform(tensor_shape.scalar(),
                                           minval=brightness_min,
                                           maxval=brightness_max)
        brightened_image = tf.mul(flipped_image, brightness_value)
        distort_result = tf.expand_dims(brightened_image, 0, name='DistortResu
lt')

    distort_sess = tf.Session(graph=distort_graph)

    epoch = 0
    idx = 0
    while epoch < max_epochs:
        batch = []
        labels = []
        for i in range(batch_size):
            if idx + i >= len(data):
                random.shuffle(data)
                epoch += 1
                idx = 0
            image_path = data[idx + i].encode()
            if should_distort:
                val = distort_sess.run(distort_result,
                                       feed_dict={jpeg_name: image_path})
            else:
                val = distort_sess.run(resized_image,
                                       feed_dict={jpeg_name: image_path})
            if b'dog' in ntpath.basename(image_path):
                labels.append(1)
            else:
                labels.append(0)
            batch.append(val)
        idx += batch_size
        yield batch, labels
```

In [35]:

```
sess.run(init)
```

Quick save of our model to view later

In [36]:

```
writer = tf.train.SummaryWriter('tensorboard/alexnet_retrain', graph=graph)
writer.close()
```

Train our model!

In [37]:

```
for data_batch, label_batch in get_batch(32, train_data, 1, should_distort=Tru
e):
    data_batch = np.squeeze(data_batch)
    feed_dict = {x: data_batch, labels: label_batch}
    err, acc, step, _ = sess.run([loss, accuracy, inc_step, train],
                                feed_dict=feed_dict)
    if step % 50 == 0:
        print("Step: {}\t Accuracy: {}\t Error: {}".format(step, acc, err))
```

```
Step: 50         Accuracy: 0.9375        Error: 0.0700903013349
Step: 100        Accuracy: 0.875         Error: 0.175362020731
Step: 150        Accuracy: 0.9375        Error: 0.150483578444
Step: 200        Accuracy: 0.9375        Error: 0.149619147182
Step: 250        Accuracy: 0.90625       Error: 0.124697074294
Step: 300        Accuracy: 0.90625       Error: 0.12353708595
Step: 350        Accuracy: 0.96875       Error: 0.187275096774
Step: 400        Accuracy: 1.0   Error: 0.0302645843476
Step: 450        Accuracy: 0.96875       Error: 0.108887523413
Step: 500        Accuracy: 1.0   Error: 0.0338761284947
Step: 550        Accuracy: 0.96875       Error: 0.0903983265162
Step: 600        Accuracy: 1.0   Error: 0.0455834493041
Step: 650        Accuracy: 0.90625       Error: 0.178182154894
Step: 700        Accuracy: 1.0   Error: 0.0333553180099
Step: 750        Accuracy: 0.90625       Error: 0.149356365204
Step: 800        Accuracy: 0.90625       Error: 0.187900155783
Step: 850        Accuracy: 0.96875       Error: 0.0794009119272
Step: 900        Accuracy: 1.0   Error: 0.0737376660109
Step: 950        Accuracy: 0.96875       Error: 0.148608088493
Step: 1000       Accuracy: 0.9375        Error: 0.0959873497486
Step: 1050       Accuracy: 0.9375        Error: 0.279115825891
```

Validate

In [38]:

```
def check_accuracy(valid_data):
    batch_size = 50
    num_correct = 0
    total = len(valid_data)
    i = 0
    for data_batch, label_batch in get_batch(batch_size, valid_data, 1):
        feed_dict = {x: data_batch, labels: label_batch}
        correct_guesses = sess.run(correct_prediction,
                                   feed_dict=feed_dict)
        num_correct += np.sum(correct_guesses)
        i += batch_size
        if i % (batch_size * 10) == 0:
            print('\tIntermediate accuracy: {}'.format((float(num_correct) / f
loat(i))))
    acc = num_correct / float(total)
    print('\nAccuracy: {}'.format(acc))
```

In [39]:
```
check_accuracy(valid_data)
```

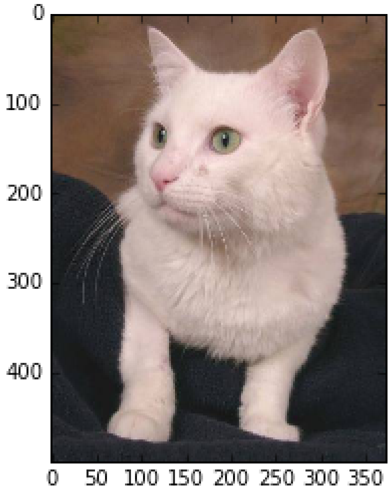
```
        Intermediate accuracy: 0.96
        Intermediate accuracy: 0.944
        Intermediate accuracy: 0.939333333333
        Intermediate accuracy: 0.9445
        Intermediate accuracy: 0.9448
        Intermediate accuracy: 0.945333333333
        Intermediate accuracy: 0.946285714286
        Intermediate accuracy: 0.945
        Intermediate accuracy: 0.945555555556
        Intermediate accuracy: 0.9452
        Intermediate accuracy: 0.944727272727
        Intermediate accuracy: 0.945666666667
        Intermediate accuracy: 0.945846153846
        Intermediate accuracy: 0.947142857143
        Intermediate accuracy: 0.946533333333

Accuracy: 0.953066666667
```

Once again, let's inspect for fun

In [40]:
```
def spot_check():
    filename = random.choice(valid_data)
    image = ndimage.imread(filename)
    feed_dict = {x: [image]}
    guess = sess.run(label_prediction, feed_dict=feed_dict)
    if guess[0] == 1:
        print('Guess: dog')
    else:
        print('Guess: cat')
    plt.imshow(image)
    plt.show()
```

In [43]:
```
spot_check()
```

```
Guess: cat
```



In [ ]: