Using AlexNet to get Encoded Vectors for Image Retrieval

In [1]:
```
import random
import tensorflow as tf
import numpy as np
import os
from scipy import ndimage
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

%matplotlib inline
```

Import graph as before

In [2]:
```
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')
```

We still want handle to `fc7`, but we won't attach anything else

In [3]:
```
# Get outputs from second-to-last layer in pre-built model
fc7_op = graph.get_operation_by_name('fc7/relu')
fc7 = fc7_op.outputs[0]
x = graph.get_operation_by_name('input').outputs[0]
init = graph.get_operation_by_name('init')

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

In [4]:
```
print(fc7.get_shape()[1])
```

```
4096
```

Get data, as before

In [5]:
```
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
```

In [6]:
```
random.shuffle(all_files)
```

Decide how many examples want for our nearest neighbors model

In [7]:
```
num_images = 5000
neighbor_list = all_files[:num_images]
```

Create empty NumPy array to fill with encoded vectors

In [8]:
```
extracted_features = np.ndarray((num_images, fc7.get_shape()[1]))
```

Fill said NumPy array

In [9]:
```
for i, filename in enumerate(neighbor_list):
    image = ndimage.imread(filename)
    features = sess.run(fc7, feed_dict={x: [image]})
    extracted_features[i:i+1] = features
    if i % 250 == 0:
        print(i)
```

```
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
2750
3000
3250
3500
3750
4000
4250
4500
4750
```

In [10]:
```
len(extracted_features)
```

Out[10]: 5000

Create Nearest Neighbors model!

In [11]:
```
nbrs = NearestNeighbors(n_neighbors=5, algorithm='ball_tree').fit(extracted_featu
```

In [12]:
```
distances, indices = nbrs.kneighbors(extracted_features)
```

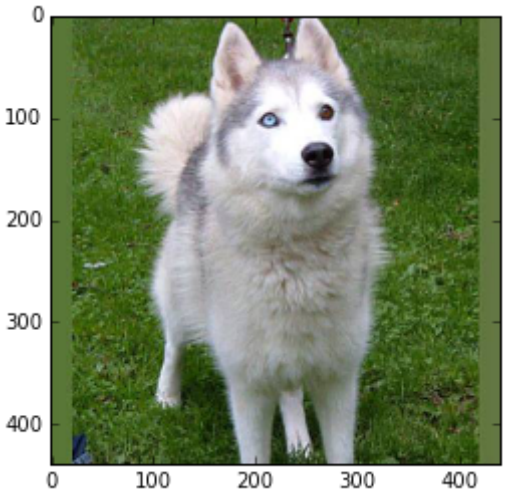
In [13]:
```
indices
```

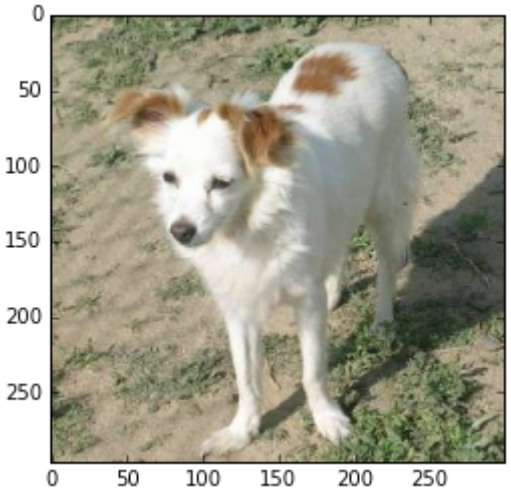
Out[13]:
```
array([[   0,   55, 1980, 3333,  396],
       [   1, 3333,  492, 4640, 1572],
       [   2,  854, 4388,  426, 3575],
       ...,
       [4997, 1590, 4828, 2728,  121],
       [4998, 2636, 1614, 3297,  981],
       [4999, 3439, 4785, 3672, 3727]])
```

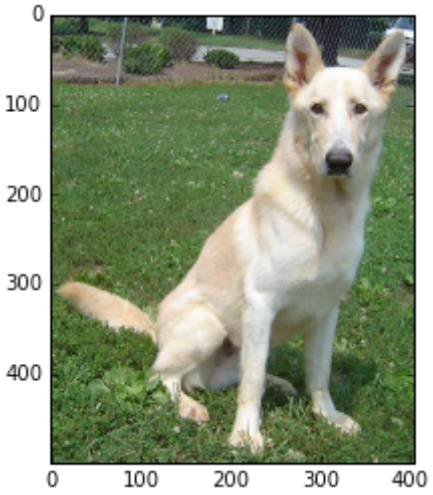
Print out the five nearest neighbors

In [14]:
```
def show_neighbors(idx, indices, filenames):
    neighbors = indices[idx]
    for i, neighbor in enumerate(neighbors):
        image = ndimage.imread(filenames[neighbor])
        plt.figure(i)
        plt.imshow(image)
    plt.show()
```

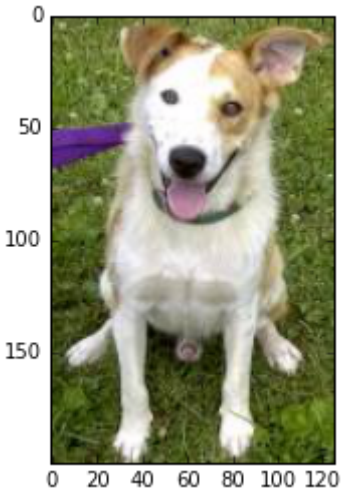
In [18]:
```
show_neighbors(random.randint(5, len(extracted_features)), indices, neighbor_list
```








0
50
100
150
200
0
50
100
150
200
250
300

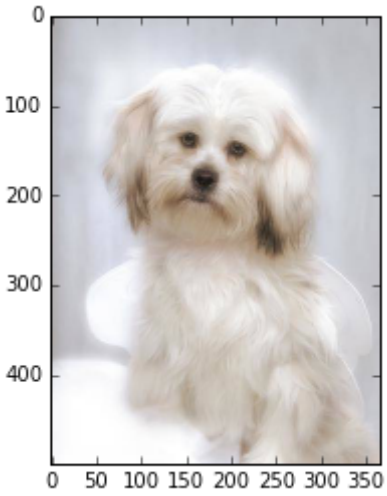

0
50
100
150
0
20
40
60
80
100
120

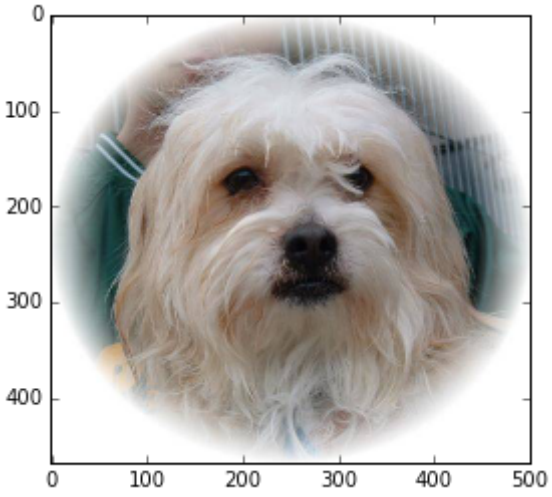
In [16]:
```
show_neighbors(random.randint(8, len(extracted_features)), indices, neighbor_list
```

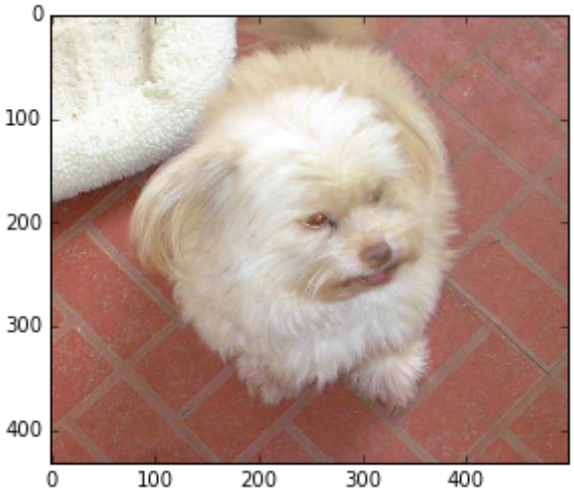

0
100
200
300
400
0 50 100 150 200 250 300

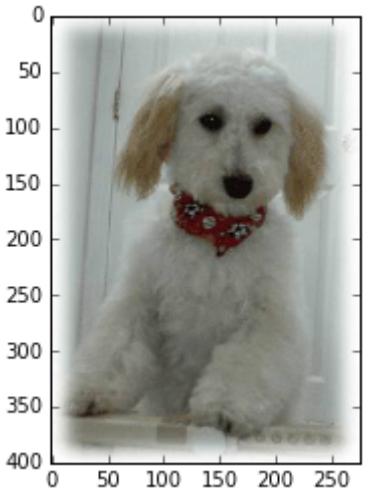

0
50
100
150
200
250
300
350
0 100 200 300 400

In [19]:
```
show_neighbors(random.randint(9, len(extracted_features)), indices, neighbor_list
```

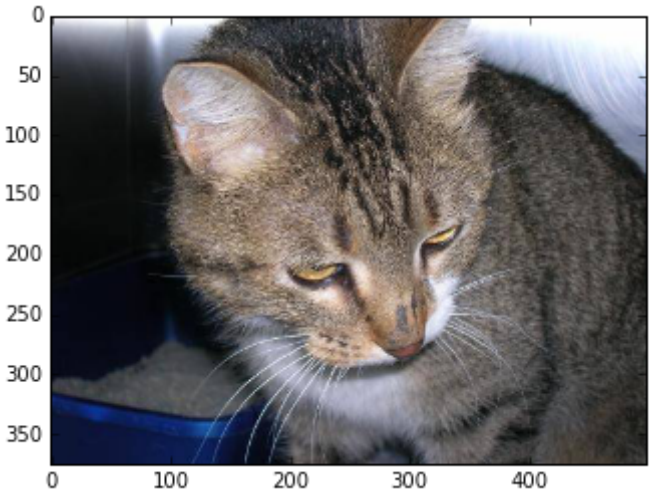


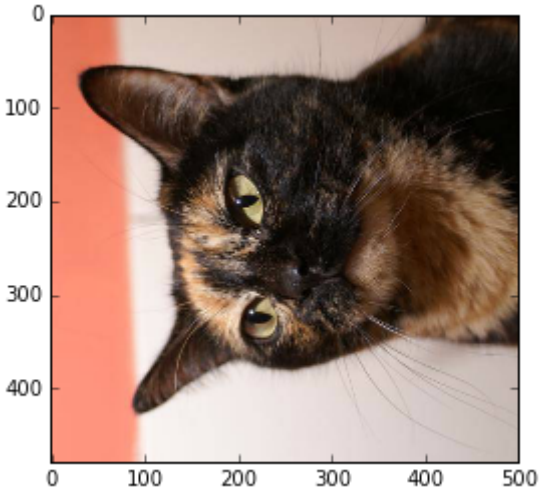


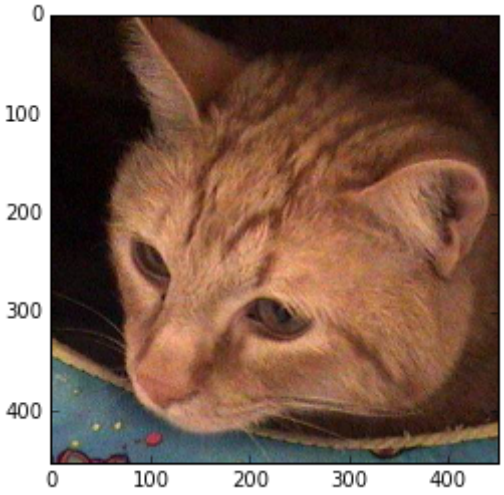

0
100
200
300
400
0
100
200
300
400

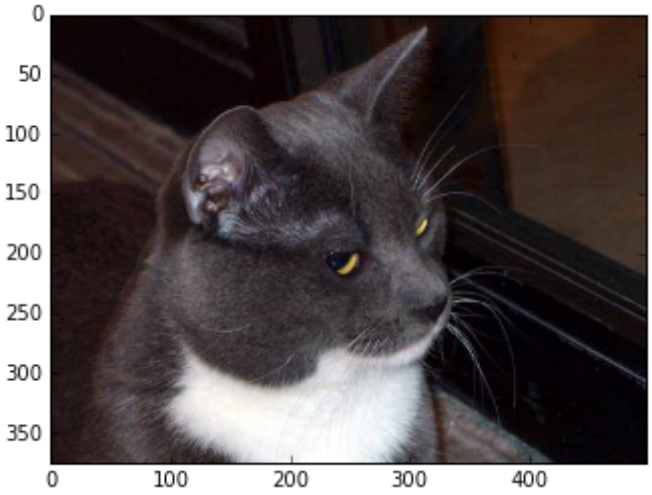

0
50
100
150
200
250
300
350
400
0
50
100
150
200
250

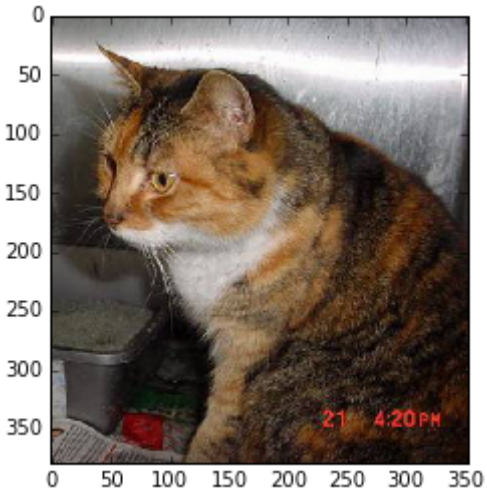
In [33]:
```
show_neighbors(random.randint(4, len(extracted_features)), indices, neighbor_list
```








0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
21 4:20PM

In [ ]: