In [1]:
```
%matplotlib inline
```

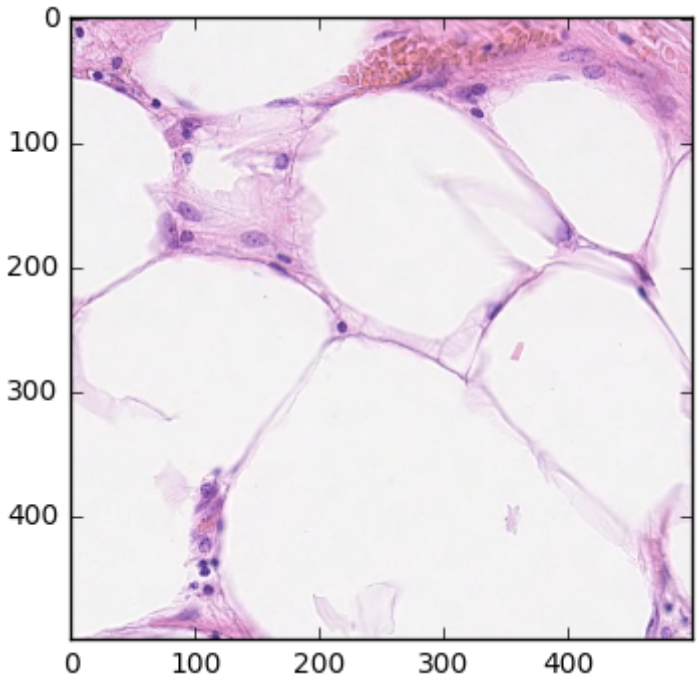
In [118]:
```
import tensorflow as tf
import tensorflow.contrib.slim as slim
from scipy.io import loadmat
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import os
import itertools
import math
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
```

In [351]:
```
from classifier_utils import get_image_data
from classifier_utils import draw
from classifier_utils import get_dataset
from classifier_utils import get_examples
from classifier_utils import expand_training_data
from classifier_utils import get_accuracy, get_weighted_f1, get_confusion
```

In [3]:
```
i = mpimg.imread('Dataset/CRCHistoPhenotypes_2016_04_28/Classification/img1/img1.bmp')
```

In [4]:
```
plt.imshow(i)
```

Out[4]: <matplotlib.image.AxesImage at 0x7f15d237e050>



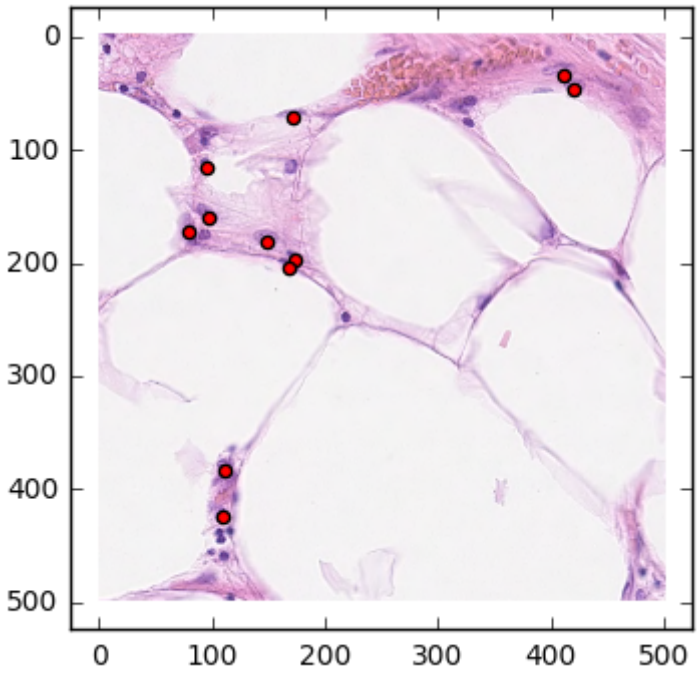
In [5]:
```
d = loadmat('Dataset/CRCHistoPhenotypes_2016_04_28/Classification/img1/img1_fibroblast.mat')['detection']
```

In [6]:
```
d
```

Out[6]:
```
array([[ 148.78269485,  181.47754293],
       [  96.60303831,  159.68097754],
       [  79.42998679,  172.89101717],
       [ 171.9002642 ,   71.834214  ],
       [  95.77741083,  116.08784676],
       [ 410.34147952,   33.19484808],
       [ 419.91875826,   46.73513871],
       [ 172.89101717,  197.16446499],
       [ 168.2675033 ,  204.42998679],
       [ 110.96895641,  383.42602378],
       [ 108.65719947,  424.70739762]])
```

In [7]:
```
plt.imshow(i)
plt.scatter(d[:,0], d[:,1], c='r')
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x7f15d21fcb90>
```



In [8]:
```
os.path.splitext("Dataset/CRCHistoPhenotypes_2016_04_28/whatever.bmp")
```

Out[8]:
```
('Dataset/CRCHistoPhenotypes_2016_04_28/whatever', '.bmp')
```

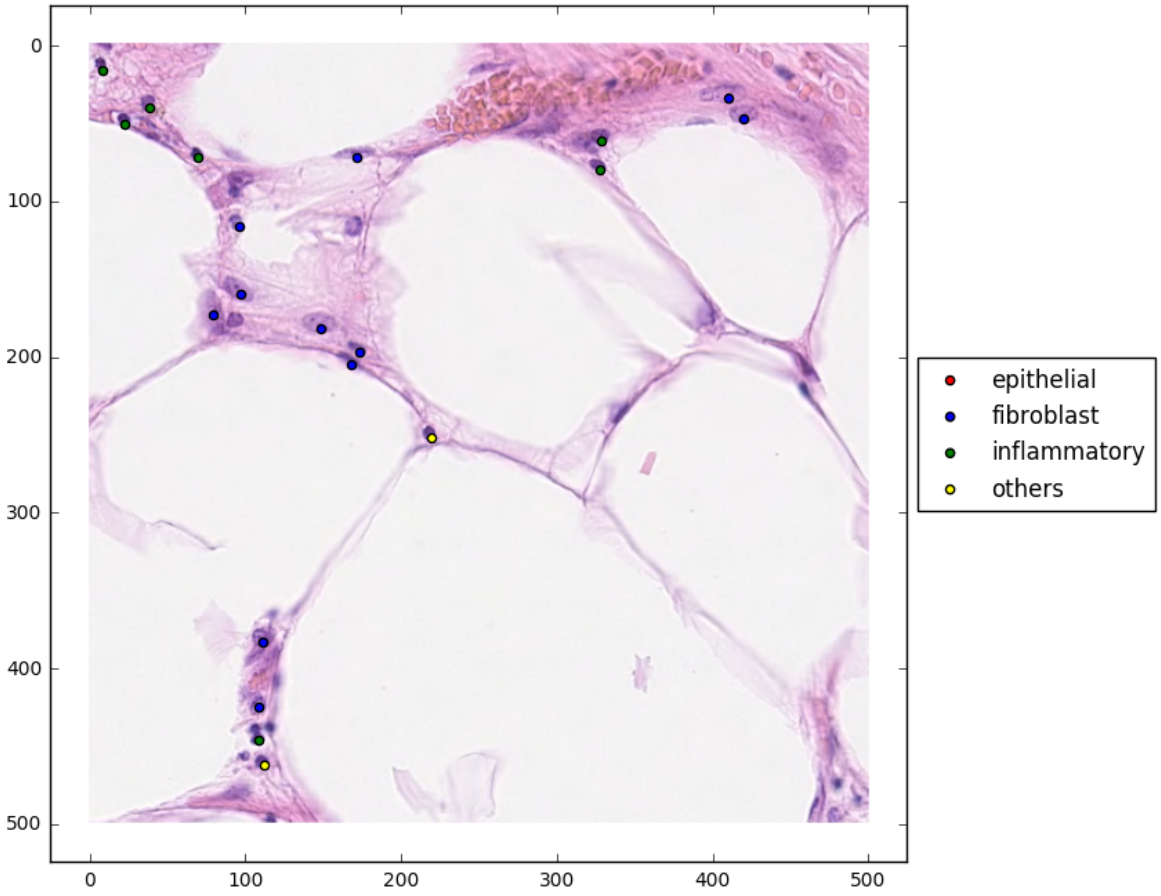
In [9]:
```
categories = [
    'epithelial',
    'fibroblast',
    'inflammatory',
    'others',
]
```

In [339]:
```
(img, centres, labels) = get_image_data('Dataset/CRCHistoPhenotypes_2016_
04_28/Classification/img1/img1.bmp', categories)
```

In [12]:
```
np.hstack((centres, labels)).astype('int32')
```

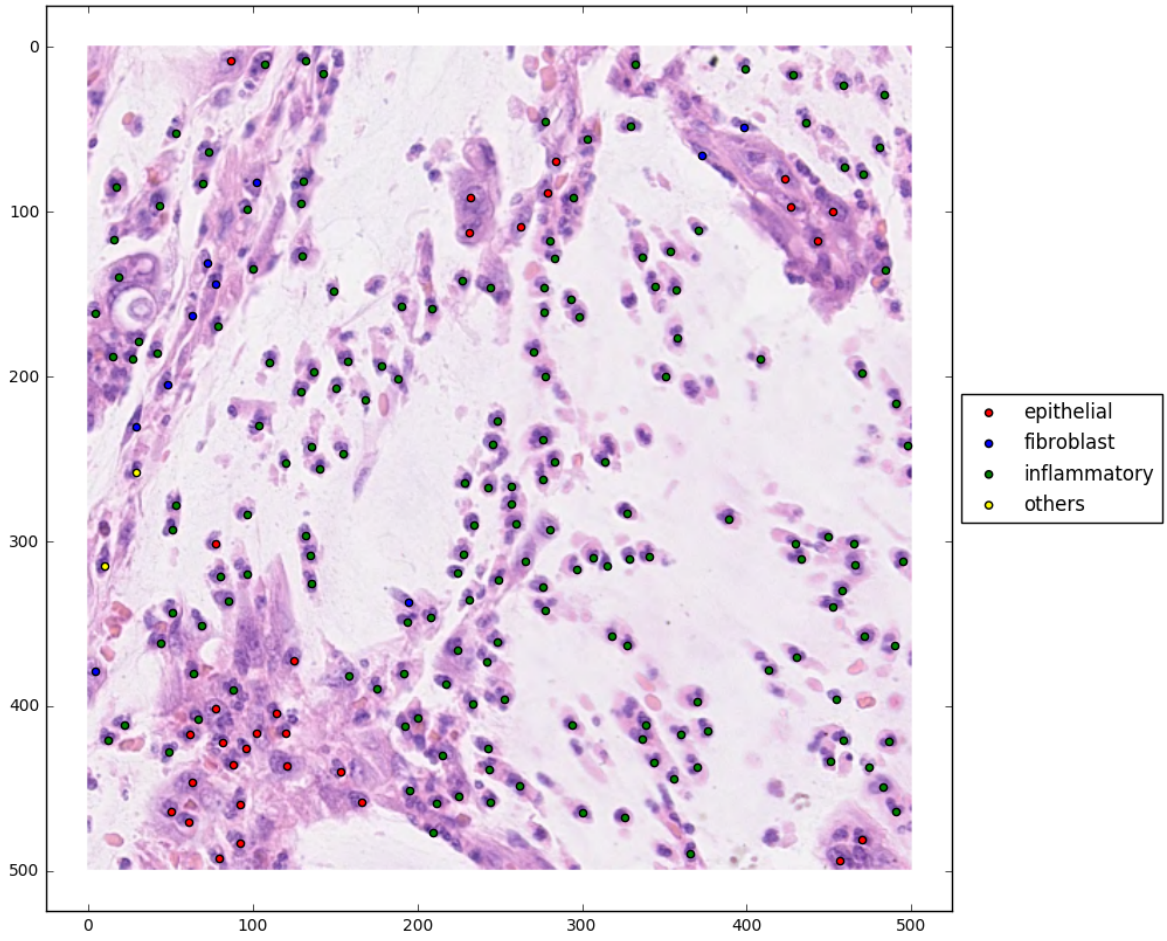
Out[12]:
```
array([[148, 181,   0,   1,   0,   0],
       [ 96, 159,   0,   1,   0,   0],
       [ 79, 172,   0,   1,   0,   0],
       [171,  71,   0,   1,   0,   0],
       [ 95, 116,   0,   1,   0,   0],
       [410,  33,   0,   1,   0,   0],
       [419,  46,   0,   1,   0,   0],
       [172, 197,   0,   1,   0,   0],
       [168, 204,   0,   1,   0,   0],
       [110, 383,   0,   1,   0,   0],
       [108, 424,   0,   1,   0,   0],
       [  8,  15,   0,   0,   1,   0],
       [ 38,  39,   0,   0,   1,   0],
       [ 22,  50,   0,   0,   1,   0],
       [ 69,  72,   0,   0,   1,   0],
       [328,  60,   0,   0,   1,   0],
       [327,  79,   0,   0,   1,   0],
       [108, 446,   0,   0,   1,   0],
       [219, 251,   0,   0,   0,   1],
       [111, 462,   0,   0,   0,   1]], dtype=int32)
```

In [340]:
```
draw(img, centres, labels, categories, figsize=(8,8))
```



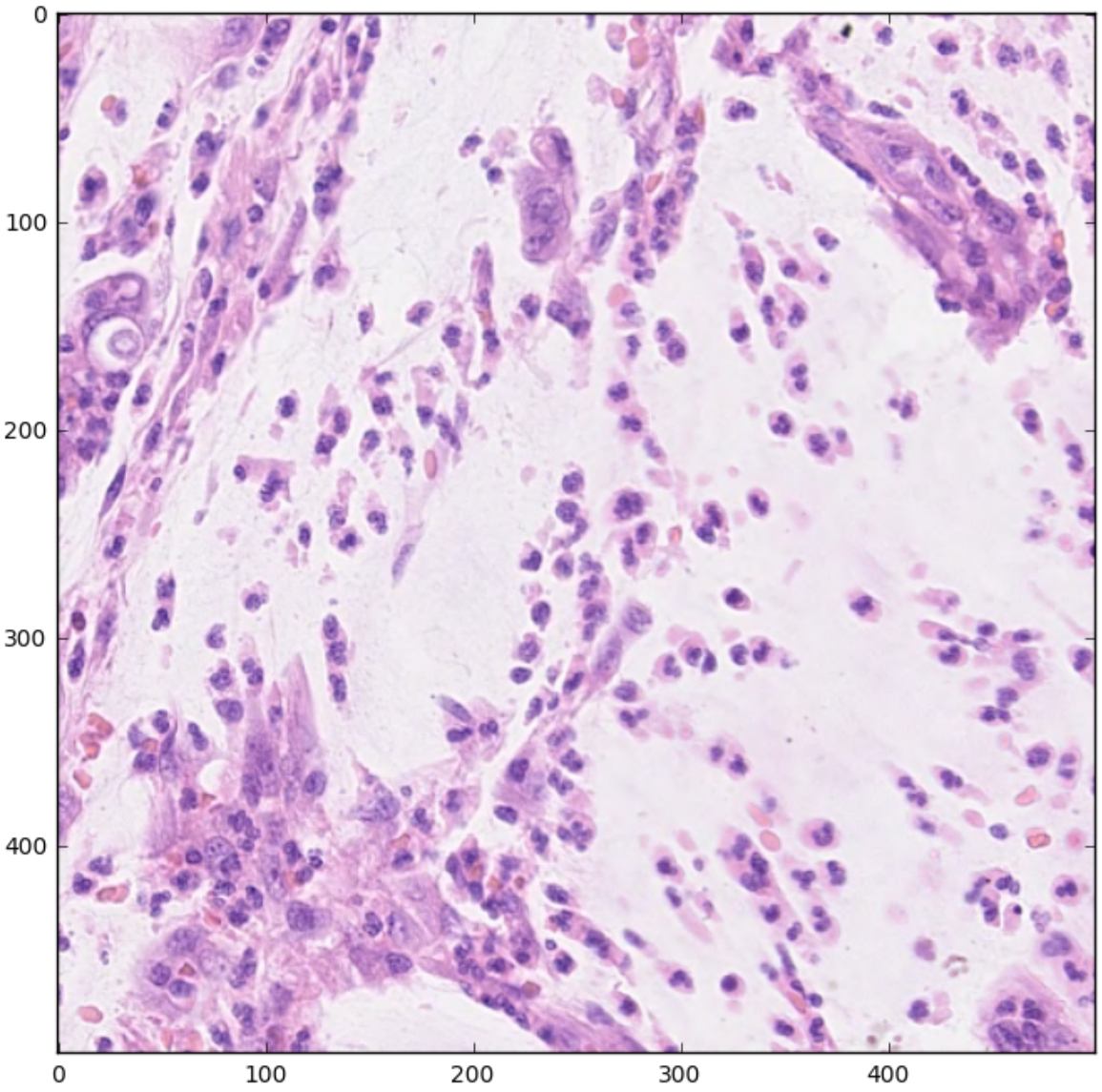
In [345]:
```
(i, c, L) = get_dataset(100, categories)
```

In [18]:
```
(img, centres, labels) = get_image_data('Dataset/CRCHistoPhenotypes_2016_
04_28/Classification/img6/img6.bmp', categories)
draw(img, centres, labels, categories)
```



In [19]:
```
plt.figure(figsize=(8,8))
plt.imshow(img)
```

Out[19]: <matplotlib.image.AxesImage at 0x7f15cb9b0fd0>



In [348]:
```
(patches, labels, centres, img_ids) = get_examples(i, c, L, 27, 27)
```

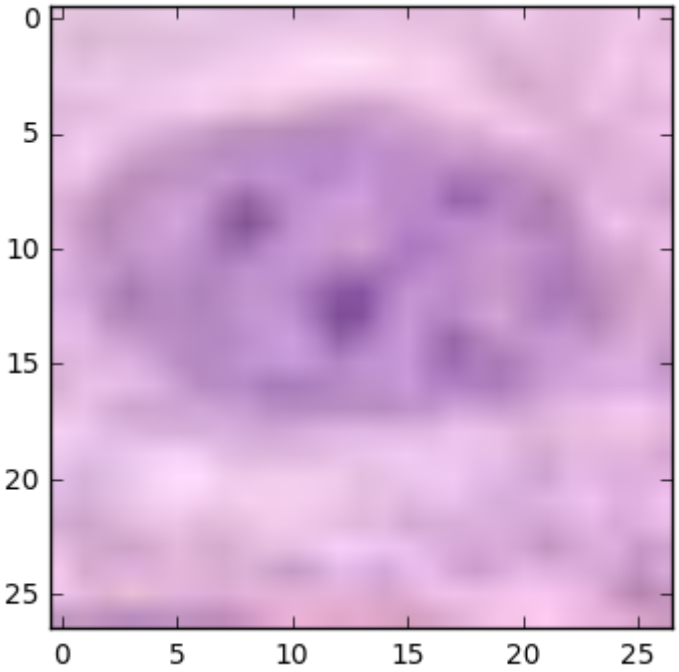
Dropped 2082 patches because too close to image border

In [22]:
```
patches.shape, labels.shape, centres.shape, img_ids.shape
```

Out[22]: ((20362, 27, 27, 3), (20362, 4), (20362, 2), (20362,))

In [24]:
```
test_index = 0
(x, y) = centres[test_index]
plt.imshow(i[img_ids[test_index]][y-13:y+14,x-13:x+14,:])
print np.all(i[img_ids[test_index]][y-13:y+14,x-13:x+14,:] == patches[tes
t_index])
```

```
True
```



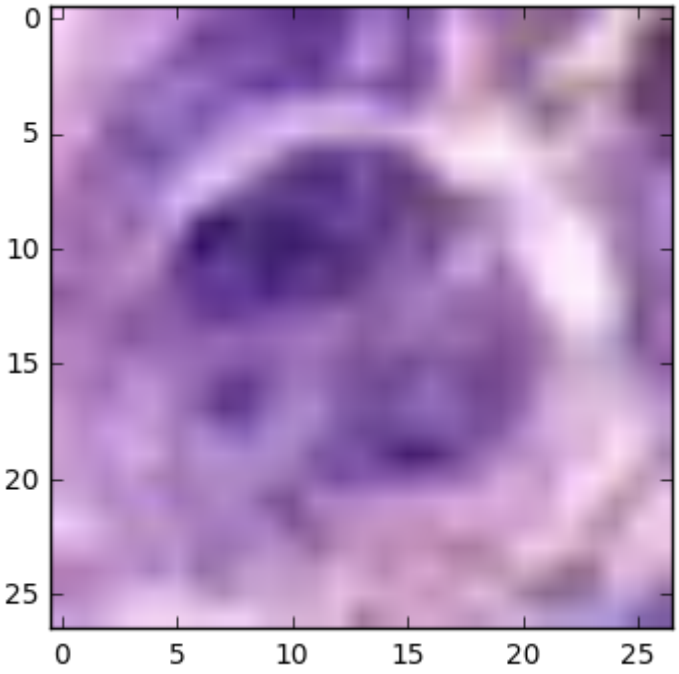
In [25]:
```
patches.shape, labels.shape, centres.shape, img_ids.shape

N = patches.shape[0]
np.random.seed(0) # predictable shuffling for now
perm = np.random.permutation(N)
patches_shuffled = patches[perm]
labels_shuffled = labels[perm]
centres_shuffled = centres[perm]
img_ids_shuffled = img_ids[perm]

num_train = int(0.8 * N)
train_patches = patches_shuffled[:num_train]
train_labels = labels_shuffled[:num_train]
train_centres = centres_shuffled[:num_train]
train_img_ids = img_ids_shuffled[:num_train]

test_patches = patches_shuffled[num_train:]
test_labels = labels_shuffled[num_train:]
test_centres = centres_shuffled[num_train:]
test_img_ids = img_ids_shuffled[num_train:]
```

In [26]:
```
train_patches.shape, train_labels.shape, test_patches.shape,
test_labels.shape
```

Out[26]: ((16289, 27, 27, 3), (16289, 4), (4073, 27, 27, 3), (4073, 4))

In [27]:
```
plt.imshow(train_patches[0])
print categories[np.argmax(train_labels[0])]
```

```
inflammatory
```



In [28]:
```
np.sum(train_labels, axis=0)
```

Out[28]:
```
array([5683, 4074, 5043, 1489])
```

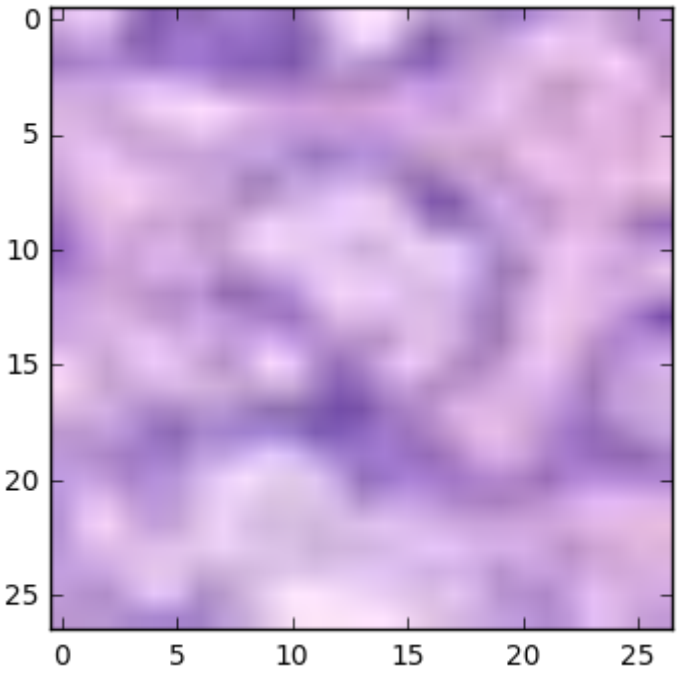
In [277]:
```
try:
    del train_dict # if it exists
    del sorted_train_dict # if it exists
except:
    pass
sorted_train_dict = expand_training_data(i, train_patches, train_labels,
train_centres, train_img_ids, 15000)
```

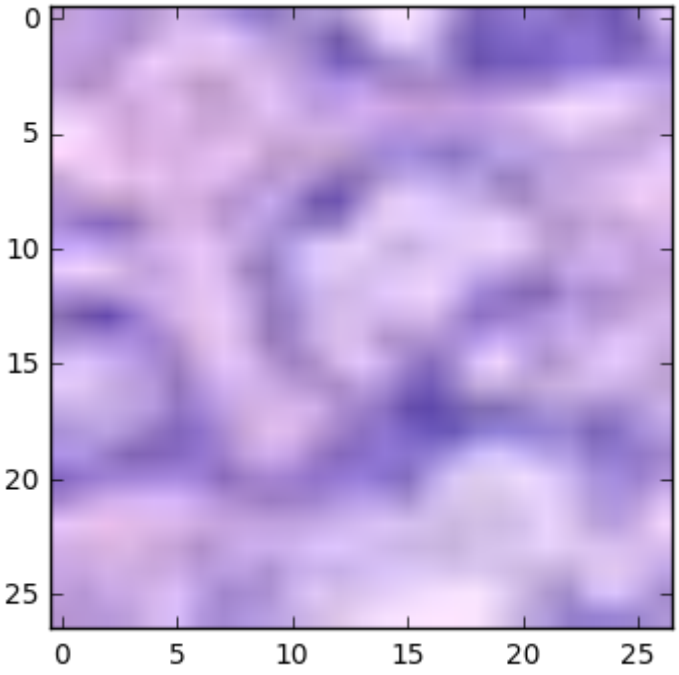
In [278]:
```
for (k, v) in sorted_train_dict.iteritems():
    print k, v.shape
```

```
hsv_factors (60000, 3)
deltas (60000, 2)
patches (60000, 27, 27, 3)
centres (60000, 2)
img_ids (60000,)
rots (60000,)
labels (60000, 4)
flips (60000,)
```

In [279]:
```
plt.imshow(train_patches[5])
plt.figure()
plt.imshow(sorted_train_dict['patches'][1])

print sorted_train_dict['flips'][1], sorted_train_dict['rots'][1], sorted
_train_dict['deltas'][1]

#print train_patches[4,:3,:3,:]
#print sorted_train_dict['patches'][0,:3,:3,:]
```

```
True 180 [2 0]
```





In [281]:
```
N = sorted_train_dict['patches'].shape[0]
np.random.seed(123) # predictable shuffling for now
perm = np.random.permutation(N)

train_dict = {k : v[perm] for (k, v) in sorted_train_dict.iteritems()}
```

In [283]:
```
sess= None
```

In [284]:
```
def classifier_model(image_batch):
    with slim.arg_scope([slim.conv2d, slim.fully_connected],
                        activation_fn=tf.nn.relu,
                        weights_initializer=tf.truncated_normal_initializ
er(stddev=0.01),
                        biases_initializer=tf.zeros_initializer,
                        weights_regularizer=slim.l2_regularizer(0.0005),
                        biases_regularizer=None):
        with slim.arg_scope([slim.conv2d, slim.max_pool2d], padding='VALI
D'):
            with slim.arg_scope([slim.dropout], keep_prob=0.8):
                net = image_batch
                net = slim.conv2d(net, 36, [4, 4], scope='1_conv')
                net = slim.max_pool2d(net, [2, 2], scope='2_max_pool')
                net = slim.conv2d(net, 48, [3, 3], scope='3_conv')
                net = slim.max_pool2d(net, [2, 2], scope='4_max_pool')
                net = slim.flatten(net, scope='4_flatten')
                net = slim.fully_connected(net, 512, scope='5_fc')
                net = slim.dropout(net, scope='5_dropout')
                net = slim.fully_connected(net, 512, scope='6_fc')
                net = slim.dropout(net, scope='6_dropout')
                net = slim.fully_connected(net, 4, activation_fn=None, sc
ope='7_fc')
                return net
```

In [285]:
```
if sess is not None:
    sess.close()
tf.reset_default_graph()
sess = tf.InteractiveSession()
patch_tensor = tf.placeholder(dtype='float32', shape=(None, 27, 27, 3))
label_tensor = tf.placeholder(dtype='int32', shape=(None, 4))
with tf.variable_scope("classifier"):
    with slim.arg_scope([slim.dropout], is_training=True):
        prediction_logits = classifier_model(patch_tensor)
with tf.variable_scope("classifier", reuse=True):
    with slim.arg_scope([slim.dropout], is_training=False):
        inference_prediction_logits = classifier_model(patch_tensor)
```

In [286]:
```
def get_training_op(prediction_logits, labels):
    loss = slim.losses.softmax_cross_entropy(prediction_logits, labels)
    total_loss = slim.losses.get_total_loss()
    optimizer = tf.train.MomentumOptimizer(momentum=0.9, learning_rate=5e
-4)#learning_rate=0.01)
    #optimizer = tf.train.AdamOptimizer()
    train_op = slim.learning.create_train_op(total_loss, optimizer)
    return (train_op, loss)
```

In [287]:
```
(train_op, loss) = get_training_op(prediction_logits, label_tensor)
```

In [288]:
```
predictions = slim.softmax(prediction_logits)
inference_predictions = slim.softmax(inference_prediction_logits)
```

In [289]:
```
predictions.get_shape(), inference_predictions.get_shape(),
label_tensor.get_shape()
```

Out[289]:
```
(TensorShape([Dimension(None), Dimension(4)]),
 TensorShape([Dimension(None), Dimension(4)]),
 TensorShape([Dimension(None), Dimension(4)]))
```

In [293]:
```
dropout_accuracy = get_accuracy(predictions, label_tensor)
accuracy = get_accuracy(inference_predictions, label_tensor)
f1 = get_weighted_f1(inference_predictions, label_tensor)
confusion = get_confusion(inference_predictions, label_tensor)
```

In [311]:
```
def train_loop(sess, patch_tensor, label_tensor, train_patches, train_lab
els, test_patches, test_labels, train_op, softmax_loss, epochs, batch_siz
e, predictions, dropout_accuracy, accuracy, f1, confusion, reset=True):
    tr_loss = []
    tst_loss = []
    N = train_patches.shape[0]
    assert train_labels.shape[0] == N
    if reset:
        sess.run(tf.initialize_all_variables())
    for e in xrange(epochs):
        for i in xrange(0, N, batch_size):
            [_, loss, p] = sess.run([train_op, softmax_loss,
predictions], feed_dict={
                    patch_tensor:train_patches[i:i+batch_size],
                    label_tensor:train_labels[i:i+batch_size],
                })
            step = i / batch_size
            if step % 25 == 0:
                [test_loss, dacc, acc, f] = sess.run([softmax_loss, dropo
ut_accuracy, accuracy, f1], feed_dict={
                    patch_tensor:test_patches,
                    label_tensor:test_labels,
                })
                print "Epoch %d, step %d, training loss %f, test_loss %f,
 accuracy = %f/%f, f1 = %f" % (e, step, loss, test_loss, dacc, acc, f)
                tr_loss.append(loss)
                tst_loss.append(test_loss)
        [test_loss, dacc, acc, f, conf] = sess.run([softmax_loss, dropout
_accuracy, accuracy, f1, confusion], feed_dict={
            patch_tensor:test_patches,
            label_tensor:test_labels,
        })
        print "End of epoch %d, training loss %f, test_loss %f, accuracy
 = %f/%f, f1 = %f" % (e, loss, test_loss, dacc, acc, f)
        print "Confusion matrix:"
        print conf
    return (tr_loss, tst_loss)
```

In [295]:
```
# no augmentation
#tr_loss, tst_loss = train_loop(sess, patch_tensor, label_tensor, train_p
atches, train_labels, test_patches, test_labels, train_op, loss, 20, 100,
 predictions, dropout_accuracy, accuracy, f1, confusion)
# with augmentation
tr_loss, tst_loss = train_loop(sess, patch_tensor, label_tensor, train_di
ct['patches'], train_dict['labels'], test_patches, test_labels, train_op,
 loss, 10, 100, predictions, dropout_accuracy, accuracy, f1, confusion)
```

```
Epoch 9, step 225, training loss 0.861277, test_loss 0.748208, accuracy =
 0.715198/0.722072, f1 = 0.722154
Epoch 9, step 250, training loss 0.857543, test_loss 0.749148, accuracy =
 0.714461/0.726737, f1 = 0.729641
Epoch 9, step 275, training loss 0.827899, test_loss 0.767274, accuracy =
 0.705868/0.714952, f1 = 0.721258
Epoch 9, step 300, training loss 0.827952, test_loss 0.783552, accuracy =
 0.691628/0.700221, f1 = 0.702412
Epoch 9, step 325, training loss 1.056151, test_loss 0.782347, accuracy =
 0.697520/0.701694, f1 = 0.705863
Epoch 9, step 350, training loss 0.842201, test_loss 0.756889, accuracy =
 0.711024/0.716916, f1 = 0.722787
Epoch 9, step 375, training loss 0.820612, test_loss 0.750400, accuracy =
 0.714461/0.720108, f1 = 0.723351
Epoch 9, step 400, training loss 0.870176, test_loss 0.764686, accuracy =
 0.703167/0.710042, f1 = 0.717933
Epoch 9, step 425, training loss 0.905569, test_loss 0.699593, accuracy =
 0.731156/0.732875, f1 = 0.723545
Epoch 9, step 450, training loss 0.883726, test_loss 0.779491, accuracy =
 0.691137/0.703413, f1 = 0.714516
Epoch 9, step 475, training loss 0.693343, test_loss 0.729867, accuracy =
 0.717653/0.726000, f1 = 0.722386
Epoch 9, step 500, training loss 0.793699, test_loss 0.789056, accuracy =
 0.696784/0.705377, f1 = 0.712449
Epoch 9, step 525, training loss 0.998048, test_loss 0.763856, accuracy =
 0.706113/0.713725, f1 = 0.716686
Epoch 9, step 550, training loss 0.901893, test_loss 0.820946, accuracy =
 0.670513/0.682789, f1 = 0.698408
Epoch 9, step 575, training loss 0.866222, test_loss 0.782978, accuracy =
 0.702922/0.711760, f1 = 0.715125
End of epoch 9, training loss 0.828047, test_loss 0.724290, accuracy = 0.
731647/0.732138, f1 = 0.732246
Confusion matrix:
[[1083  170  105   35]
 [ 129  705  163   72]
 [  31  100 1035   87]
 [  20   72  107  159]]
```

In [312]:
```
# Try another 10 epochs
tr_loss2, tst_loss2 = train_loop(sess, patch_tensor, label_tensor, train_
dict['patches'], train_dict['labels'], test_patches, test_labels, train_o
p, loss, 10, 100, predictions, dropout_accuracy, accuracy, f1, confusion,
 reset=False)
```

```
Epoch 9, step 225, training loss 0.665841, test_loss 0.743780, accuracy =
 0.714461/0.719863, f1 = 0.728241
Epoch 9, step 250, training loss 0.688977, test_loss 0.717920, accuracy =
 0.730174/0.744905, f1 = 0.744861
Epoch 9, step 275, training loss 0.716597, test_loss 0.746891, accuracy =
 0.717407/0.729683, f1 = 0.729617
Epoch 9, step 300, training loss 0.663885, test_loss 0.706045, accuracy =
 0.733366/0.741959, f1 = 0.741381
Epoch 9, step 325, training loss 0.845116, test_loss 0.738857, accuracy =
 0.723300/0.736067, f1 = 0.737651
Epoch 9, step 350, training loss 0.638910, test_loss 0.733105, accuracy =
 0.716916/0.724282, f1 = 0.732080
Epoch 9, step 375, training loss 0.704500, test_loss 0.758284, accuracy =
 0.715689/0.722809, f1 = 0.728328
Epoch 9, step 400, training loss 0.701134, test_loss 0.720230, accuracy =
 0.717898/0.735085, f1 = 0.743507
Epoch 9, step 425, training loss 0.779204, test_loss 0.670482, accuracy =
 0.748588/0.750307, f1 = 0.746413
Epoch 9, step 450, training loss 0.680333, test_loss 0.744098, accuracy =
 0.712497/0.726246, f1 = 0.732868
Epoch 9, step 475, training loss 0.580466, test_loss 0.695251, accuracy =
 0.743432/0.755463, f1 = 0.756033
Epoch 9, step 500, training loss 0.601554, test_loss 0.722973, accuracy =
 0.723300/0.732384, f1 = 0.737698
Epoch 9, step 525, training loss 0.744669, test_loss 0.680748, accuracy =
 0.745151/0.750798, f1 = 0.751176
Epoch 9, step 550, training loss 0.738777, test_loss 0.724834, accuracy =
 0.724527/0.741714, f1 = 0.748154
Epoch 9, step 575, training loss 0.687487, test_loss 0.776052, accuracy =
 0.709796/0.718144, f1 = 0.719238
End of epoch 9, training loss 0.552169, test_loss 0.693133, accuracy = 0.
742941/0.750552, f1 = 0.751103
Confusion matrix:
[[1090  206   76   21]
 [ 114  793  112   50]
 [  46  130 1002   75]
 [  23   75   88  172]]
```

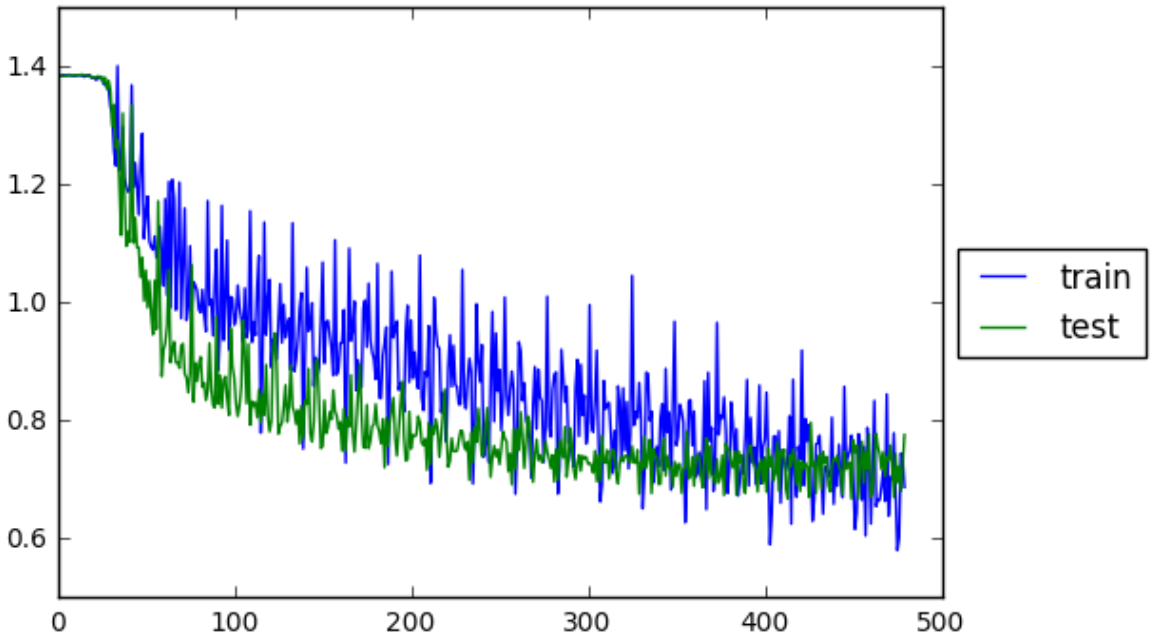
In [313]:
```
# Save the model
saver = tf.train.Saver()
save_path = saver.save(sess, "classification_models/v3_augmentation/mode
l.ckpt")
print "Saved to:", save_path
```

```
Saved to: classification_models/v3_augmentation/model.ckpt
```

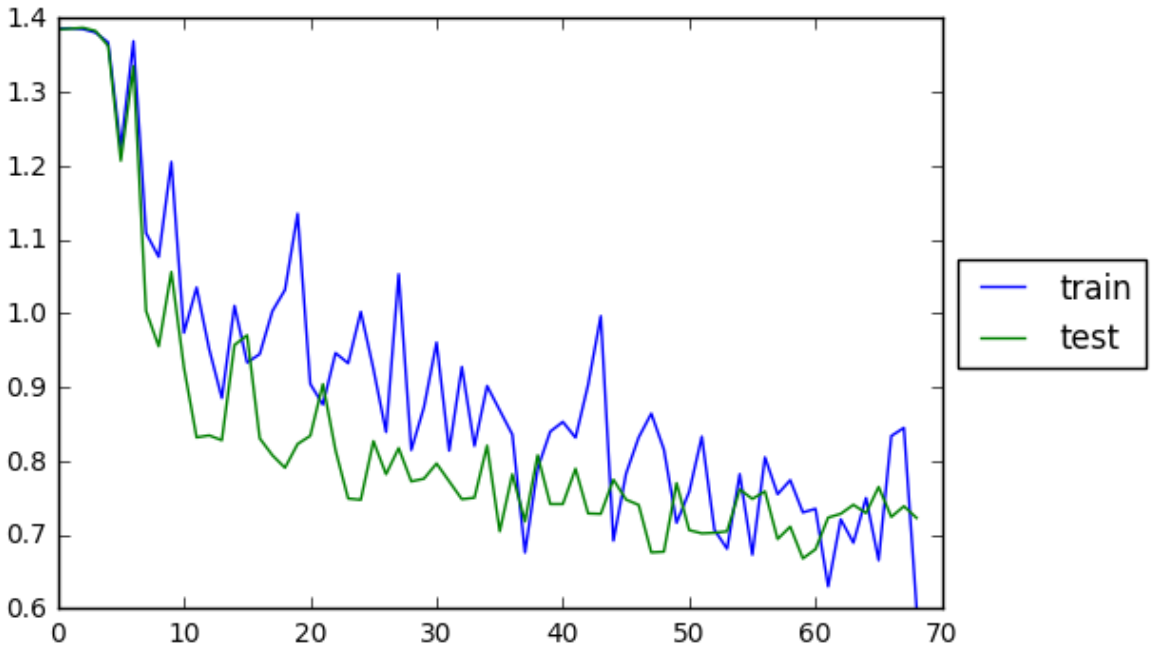
In [ ]:
```
# Restore the model
saver = tf.train.Saver()
saver.restore(sess, "classification_models/v1/model.ckpt")
```

In [314]:
```
x = xrange(len(tr_loss + tr_loss2))
plt.plot(x, tr_loss + tr_loss2, label='train')
plt.plot(x, tst_loss + tst_loss2, label='test')
plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
```

Out[314]: <matplotlib.legend.Legend at 0x7f15acde1ed0>



In [315]:
```
@interact(s=widgets.IntSlider(min=0,max=23,step=1,value=0))
def f(s):
    x = xrange(len((tr_loss + tr_loss2)[s::7]))
    plt.plot(x, (tr_loss + tr_loss2)[s::7], label='train')
    plt.plot(x, (tst_loss + tst_loss2)[s::7], label='test')
    plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
```



In [300]:
```
def get_NEP_prediction(sess, patch_tensor, prediction_tensor, img,
centre, H, W, d):
    assert H%2==1
    assert W%2==1
    halfH = (W-1)/2
    halfW = (H-1)/2
    (imgH, imgW, _) = img.shape
    (x, y) = centre

    # Helper function to check for out of bounds
    def inbounds(dx, dy):
        return (x + dx - halfW >= 0 and
                x + dx + halfW < imgW and
                y + dy - halfH >= 0 and
                y + dy + halfH < imgH)

    # Iterate over each neighbour position
    patches = []
    for dx in range(-d, d+1):
        for dy in range(-d, d+1):
            if dx**2 + dy**2 <= d**2 and inbounds(dx, dy):
                patches.append(img[y+dy-halfH:y+dy+halfW+1,x+dx-halfW:x+d
x+halfW+1,:])
                #print (dx, dy, math.sqrt(dx**2+dy**2))
    assert len(patches) > 0
    patches = np.stack(patches)
    #print patches.shape

    # Run prediction
    predictions = sess.run(prediction_tensor, feed_dict={
            patch_tensor: patches,
        })

    # Get average prediction
    #print predictions.shape
    average_predictions = np.mean(predictions, axis=0)
    #print average_predictions.shape
    return (average_predictions, patches)
```

In [316]:
```
example = 3
(avg_pred, patches_used) = get_NEP_prediction(sess,
                                              patch_tensor,
                                              inference_predictions,
                                              i[test_img_ids[example]],
                                              test_centres[example],
                                              27,
                                              27,
                                              4)
print avg_pred
print test_labels[example]
```

```
[ 0.90826339  0.03643493  0.04092112  0.01438036]
[1 0 0 0]
```

In [328]:
```
all_nep_predictions = []
d = 4
for ii in range(len(test_patches)):
    (avg_pred, _) = get_NEP_prediction(sess,
                                       patch_tensor,
                                       inference_predictions,
                                       i[test_img_ids[ii]],
                                       test_centres[ii],
                                       27,
                                       27,
                                       d)
    all_nep_predictions.append(avg_pred)
all_nep_predictions = np.stack(all_nep_predictions)
```

In [329]:
```
print sess.run([accuracy, f1, confusion], feed_dict={
        inference_predictions: all_nep_predictions,
        label_tensor: test_labels,
    })
```

```
[0.76651114, 0.76562738, array([[1130,  180,   71,   12],
       [ 110,  803,  111,   45],
       [  42,  125, 1023,   63],
       [  24,   77,   91,  166]], dtype=int32)]
```

Results for f1-score for various d:

0 = .75110334 1 = .75696123 2 = .76257628 3 = .76513195 4 = .76562738