In [2]:

```
%matplotlib inline
```

In [3]:

```
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 [4]:

```
import classifier_utils as utils
```

In [21]:

```
import cnn_classifier
reload(cnn_classifier)
```

Out[21]:

```
<module 'cnn_classifier' from 'cnn_classifier.py'>
```

In [6]:

```
import context_classifier
```

# Get Dataset

In [7]:

```
categories = [
    'epithelial',
    'fibroblast',
    'inflammatory',
    'others',
]

train, test = utils.get_augmented_dataset_divided_per_image(categories)

# Carve out a validation set from our test set
# Split it 50/50
# Need to shuffle the test set before splitting
np.random.seed(8080) # repeatability
N = len(test['patches'])
new_N = N/2
perm = np.random.permutation(N)
validation = {}
for k in list(test.iterkeys()):
    values = test[k]
    test[k], validation[k] = np.split(values[perm], [new_N])
```

```
Dropped 1559 patches because too close to image border
Dropped 523 patches because too close to image border
```

In [8]:
```
for (k, v) in train.iteritems():
    print "train", k, v.shape
for (k, v) in test.iteritems():
    print "test", k, v.shape
for (k, v) in validation.iteritems():
    print "validation", k, v.shape
```

```
train hsv_factors (60000, 3)
train deltas (60000, 2)
train patches (60000, 27, 27, 3)
train rots (60000,)
train labels (60000, 4)
train flips (60000,)
train centres (60000, 2)
train img_ids (60000,)
test img_ids (2296,)
test labels (2296, 4)
test patches (2296, 27, 27, 3)
test centres (2296, 2)
validation img_ids (2297,)
validation labels (2297, 4)
validation patches (2297, 27, 27, 3)
validation centres (2297, 2)
```

In [9]:
```
# Sanity-check that the test and train data come from different images
test_img_ids = set(test['img_ids'])
train_img_ids = set(train['img_ids'])
print "Train:", sorted(train_img_ids)
print "Test:", sorted(test_img_ids)
print "Intersection:", sorted(train_img_ids.intersection(test_img_ids))
```

```
Train: [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 17, 18, 19, 2
0, 22, 23, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 38, 40, 41, 42, 4
3, 45, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 62, 63, 66, 6
8, 69, 71, 72, 73, 74, 76, 77, 78, 79, 80, 82, 84, 85, 86, 89, 90, 91, 9
2, 93, 94, 95, 97, 98, 99]
Test: [9, 12, 21, 25, 36, 37, 39, 44, 46, 47, 58, 64, 65, 67, 70, 81, 83,
 87, 88, 96]
Intersection: []
```

In [10]:
```
(all_imgs, _, _) = utils.get_dataset(100, categories)
```

# Train a Softmax CNN Model

In [30]:

```
try:
    sess.close()
except:
    pass # sess doesn't exist yet!

tf.reset_default_graph()
sess = tf.InteractiveSession()

with tf.device('cpu:0'):
    patch_model = cnn_classifier.SoftmaxCNNModel()
```

In [111]:

```
num_epochs = 20
batch_size = 100
tr_loss, tst_loss = patch_model.train_loop(sess,
                                           train['patches']*255,
                                           train['labels'],
                                           test['patches']*255,
                                           test['labels'],
                                           num_epochs,
                                           batch_size)
```

```
 = 0.749401/0.755933, f1 = 0.758045
Epoch 18, step 400, training loss 0.660280, test_loss 0.751994, accuracy
 = 0.707816/0.716307, f1 = 0.726868
Epoch 18, step 425, training loss 0.704016, test_loss 0.679598, accuracy
 = 0.740692/0.746571, f1 = 0.749295
Epoch 18, step 450, training loss 0.694917, test_loss 0.792748, accuracy
 = 0.687133/0.698672, f1 = 0.703823
Epoch 18, step 475, training loss 0.669648, test_loss 0.735329, accuracy
 = 0.714783/0.721315, f1 = 0.734313
Epoch 18, step 500, training loss 0.621011, test_loss 0.672338, accuracy
 = 0.747006/0.757022, f1 = 0.756487
Epoch 18, step 525, training loss 0.570678, test_loss 0.648165, accuracy
 = 0.751361/0.760941, f1 = 0.761866
Epoch 18, step 550, training loss 0.657930, test_loss 0.665239, accuracy
 = 0.743087/0.752014, f1 = 0.752115
Epoch 18, step 575, training loss 0.656160, test_loss 0.707461, accuracy
 = 0.731983/0.743087, f1 = 0.747225
End of epoch 18, training loss 0.590699, test_loss 0.670183, accuracy =
 0.747224/0.757022, f1 = 0.758579
Confusion matrix:
[[1533  228  113   37]
 [  99  830  152   66]
 [  30   82  924   86]
 [  30   82  111  190]]
Epoch 19, step 0, training loss 0.624076, test_loss 0.687994, accuracy =
 0.741999/0.748966, f1 = 0.753196
Epoch 19, step 25, training loss 0.637699, test_loss 0.683148, accuracy =
 0.740910/0.755933, f1 = 0.758815
Epoch 19, step 50, training loss 0.522555, test_loss 0.822308, accuracy =
 0.672327/0.687568, f1 = 0.709923
Epoch 19, step 75, training loss 0.661002, test_loss 0.696586, accuracy =
 0.727847/0.742216, f1 = 0.750906
Epoch 19, step 100, training loss 0.601344, test_loss 0.638100, accuracy
 = 0.763989/0.767908, f1 = 0.762443
Epoch 19, step 125, training loss 0.620895, test_loss 0.703519, accuracy
 = 0.731113/0.744394, f1 = 0.743533
Epoch 19, step 150, training loss 0.651952, test_loss 0.687114, accuracy
 = 0.738951/0.750490, f1 = 0.747628
Epoch 19, step 175, training loss 0.661939, test_loss 0.712422, accuracy
 = 0.730024/0.738297, f1 = 0.746362
Epoch 19, step 200, training loss 0.821070, test_loss 0.722728, accuracy
 = 0.712171/0.732419, f1 = 0.738938
Epoch 19, step 225, training loss 0.634190, test_loss 0.663985, accuracy
 = 0.753320/0.750490, f1 = 0.747598
Epoch 19, step 250, training loss 0.942224, test_loss 0.785472, accuracy
 = 0.697583/0.704115, f1 = 0.719067
Epoch 19, step 275, training loss 0.680060, test_loss 0.673617, accuracy
 = 0.736991/0.745265, f1 = 0.746030
Epoch 19, step 300, training loss 0.659705, test_loss 0.726089, accuracy
 = 0.720880/0.727411, f1 = 0.734538
Epoch 19, step 325, training loss 0.592237, test_loss 0.672140, accuracy
 = 0.745482/0.752667, f1 = 0.753243
Epoch 19, step 350, training loss 0.662133, test_loss 0.765232, accuracy
 = 0.711300/0.720226, f1 = 0.725260
Epoch 19, step 375, training loss 0.538620, test_loss 0.687905, accuracy
 = 0.741128/0.747442, f1 = 0.751469
Epoch 19, step 400, training loss 0.635384, test_loss 0.750034, accuracy
```

```
 = 0.709340/0.715654, f1 = 0.726903
Epoch 19, step 425, training loss 0.745215, test_loss 0.715640, accuracy
 = 0.717178/0.728500, f1 = 0.734597
Epoch 19, step 450, training loss 0.650116, test_loss 0.827763, accuracy
 = 0.675376/0.685391, f1 = 0.692462
Epoch 19, step 475, training loss 0.594753, test_loss 0.727718, accuracy
 = 0.715001/0.729371, f1 = 0.739847
Epoch 19, step 500, training loss 0.623782, test_loss 0.656714, accuracy
 = 0.751361/0.763336, f1 = 0.761429
Epoch 19, step 525, training loss 0.557512, test_loss 0.646433, accuracy
 = 0.754627/0.763989, f1 = 0.763938
Epoch 19, step 550, training loss 0.604525, test_loss 0.683967, accuracy
 = 0.735685/0.748966, f1 = 0.745262
Epoch 19, step 575, training loss 0.667358, test_loss 0.717957, accuracy
 = 0.727194/0.734378, f1 = 0.737665
End of epoch 19, training loss 0.564958, test_loss 0.667312, accuracy =
 0.750272/0.758763, f1 = 0.760798
Confusion matrix:
[[1568  201  108   34]
 [ 116  809  149   73]
 [  33   76  904  109]
 [  32   76  101  204]]
```

In [112]:

```
# Save the model
saver = tf.train.Saver(write_version=1)
save_path = saver.save(sess, "context_models/patch_models/v2/model.ckpt")
print "Saved to:", save_path
```

```
WARNING:tensorflow:*******************************************************
*
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:   `tf.train.Saver(write_version=tf.train.SaverDef.V2)
`
WARNING:tensorflow:now on by default.
WARNING:tensorflow:*******************************************************
*
Saved to: context_models/patch_models/v2/model.ckpt
```

In [31]:

```
# Restore the model
saver = tf.train.Saver()
saver.restore(sess, "context_models/patch_models/v2/model.ckpt")
```

In [33]:

```
# Check that we get the expected f1
sess.run(patch_model.f1, feed_dict={
        patch_model.patch_tensor:np.concatenate([test['patches'], validat
ion['patches']], axis=0)*255,
        patch_model.label_tensor:np.concatenate([test['labels'], validati
on['labels']], axis=0),
    })
```

Out[33]:

```
0.76079822
```

# Get Non-Augmented Dataset

In [35]:
```
(train_vanilla, _) = utils.get_dataset_divided_per_image(categories)
```

```
Dropped 1559 patches because too close to image border
Dropped 523 patches because too close to image border
```

# Apply NEP

In [36]:
```
nep_test_probabilities = context_classifier.get_all_NEP_predictions(sess,
 test, patch_model, all_imgs)
nep_validation_probabilities = context_classifier.get_all_NEP_predictions
ss, validation, patch_model, all_imgs)
```

In [37]:
```
# Okay cool, let's try applying it to everything
print sess.run(patch_model.f1, feed_dict={
        patch_model.inference_predictions: np.concatenate([nep_test_proba
bilities, nep_validation_probabilities], axis=0),
        patch_model.label_tensor: np.concatenate([test['labels'], validat
ion['labels']], axis=0),
    })
```

```
0.777003
```

# Compute Probability Weight Neighbourhoods

In [38]:
```
bin_size = 27
context = 5
test['probabilities'], test['weight_neighbourhoods'] = \
    context_classifier.compute_probability_neighbourhoods_with_NEP(sess,
test, patch_model, all_imgs, bin_size=bin_size, context_length=context)
validation['probabilities'], validation['weight_neighbourhoods'] = \
    context_classifier.compute_probability_neighbourhoods_with_NEP(sess,
validation, patch_model, all_imgs, bin_size=bin_size, context_length=cont
ext)
train_vanilla['probabilities'], train_vanilla['weight_neighbourhoods'] =
\
    context_classifier.compute_probability_neighbourhoods_with_NEP(sess,
train_vanilla, patch_model, all_imgs, bin_size=bin_size, context_length=c
ontext)
```

# Balance Classes

In [39]:

```
# Let's just try to balance out the classes a little
selectors = []
for c in xrange(train_vanilla['labels'].shape[1]):
    selectors.append(np.nonzero(train_vanilla['labels'][:,c])[0])
largest_class_size = max([len(s) for s in selectors])
#print largest_class_size
expanded_selectors = []
for s in selectors:
    expansion_factor = largest_class_size / len(s) + 1
    expanded_selectors.append(np.repeat(s, expansion_factor)[:largest_cla
ss_size])

np.random.seed(649) # repeatability
balanced_train_vanilla = {}
N = len(selectors) * largest_class_size
perm = np.random.permutation(N)
for (k, v) in train_vanilla.iteritems():
    new = np.concatenate([v[s] for s in expanded_selectors], axis=0)
    #print new.shape
    balanced_train_vanilla[k] = new[perm]
    #print balanced_train_vanilla[k].shape
```

In [40]:

```
for (k, v) in balanced_train_vanilla.iteritems():
    print k, v.shape
```

```
patches (20696, 27, 27, 3)
probabilities (20696, 4)
labels (20696, 4)
weight_neighbourhoods (20696, 5, 5, 4)
centres (20696, 2)
img_ids (20696,)
```

In [41]:

```
print np.count_nonzero(balanced_train_vanilla['labels'][:,0])
print np.count_nonzero(balanced_train_vanilla['labels'][:,1])
print np.count_nonzero(balanced_train_vanilla['labels'][:,2])
print np.count_nonzero(balanced_train_vanilla['labels'][:,3])
```

```
5174
5174
5174
5174
```

In [42]:

```
N = len(train_vanilla['labels'])
print np.count_nonzero(train_vanilla['labels'][:,0]) / float(N)
print np.count_nonzero(train_vanilla['labels'][:,1]) / float(N)
print np.count_nonzero(train_vanilla['labels'][:,2]) / float(N)
print np.count_nonzero(train_vanilla['labels'][:,3]) / float(N)
```

```
0.327541378654
0.253408586467
0.328112118714
0.0909379161646
```

# Train the Context Model

In [45]:

```
reload(context_classifier)
```

Out[45]:

```
<module 'context_classifier' from 'context_classifier.pyc'>
```

In [46]:

```
try:
    sess.close()
except:
    pass # sess doesn't exist yet!

tf.reset_default_graph()
sess = tf.InteractiveSession()

learning_rate = tf.Variable(0.001, trainable=False)
context_model = context_classifier.ContextModel(learning_rate=learning_ra
te, context_length=context)
```

In [60]:

```
num_epochs = 100
batch_size = 100
tr_loss, tst_loss = context_model.train_loop(
    sess,
    balanced_train_vanilla['probabilities'],
    balanced_train_vanilla['weight_neighbourhoods'],
    balanced_train_vanilla['labels'],
    validation['probabilities'],
    validation['weight_neighbourhoods'],
    validation['labels'],
    num_epochs,
    batch_size)
```

```
 = 0.720940/0.809317, f1 = 0.811924
Epoch 98, step 200, training loss 0.586675, test_loss 0.826884, accuracy
 = 0.725729/0.811058, f1 = 0.814869
End of epoch 98, training loss 0.590271, test_loss 0.832351, accuracy =
 0.714410/0.811493, f1 = 0.815125
Confusion matrix:
[[840  45  16  36]
 [ 33 458  59  50]
 [ 16  63 431  43]
 [ 11  21  40 135]]
Epoch 99, step 0, training loss 0.646528, test_loss 0.824344, accuracy =
 0.731824/0.811058, f1 = 0.814767
Epoch 99, step 25, training loss 0.629163, test_loss 0.830278, accuracy =
 0.717458/0.806704, f1 = 0.811170
Epoch 99, step 50, training loss 0.580530, test_loss 0.818100, accuracy =
 0.726165/0.811929, f1 = 0.815374
Epoch 99, step 75, training loss 0.571464, test_loss 0.835491, accuracy =
 0.722246/0.810623, f1 = 0.813814
Epoch 99, step 100, training loss 0.572175, test_loss 0.810651, accuracy
 = 0.730083/0.811493, f1 = 0.814985
Epoch 99, step 125, training loss 0.601582, test_loss 0.812367, accuracy
 = 0.733566/0.812364, f1 = 0.815438
Epoch 99, step 150, training loss 0.512074, test_loss 0.816640, accuracy
 = 0.722682/0.811929, f1 = 0.815670
Epoch 99, step 175, training loss 0.704735, test_loss 0.845530, accuracy
 = 0.717893/0.813235, f1 = 0.816159
Epoch 99, step 200, training loss 0.582866, test_loss 0.835289, accuracy
 = 0.717893/0.807575, f1 = 0.811509
End of epoch 99, training loss 0.581120, test_loss 0.834678, accuracy =
 0.725729/0.808446, f1 = 0.812158
Confusion matrix:
[[836  50  16  35]
 [ 32 462  58  48]
 [ 14  69 424  46]
 [ 15  20  37 135]]
```

In [25]:

```
# Load the best model
saver = tf.train.Saver()
saver.restore(sess, "context_models/neighbourhood_models/tmp/model.ckpt")

# Evaluate f1
print sess.run(context_model.f1, feed_dict={
        context_model.probability_tensor:test['probabilities'],
        context_model.neighbourhood_tensor:test['weight_neighbourhoods'],
        context_model.label_tensor:test['labels'],
    })

# Show confusion matrix
print sess.run(context_model.confusion, feed_dict={
        context_model.probability_tensor:test['probabilities'],
        context_model.neighbourhood_tensor:test['weight_neighbourhoods'],
        context_model.label_tensor:test['labels'],
    })
```

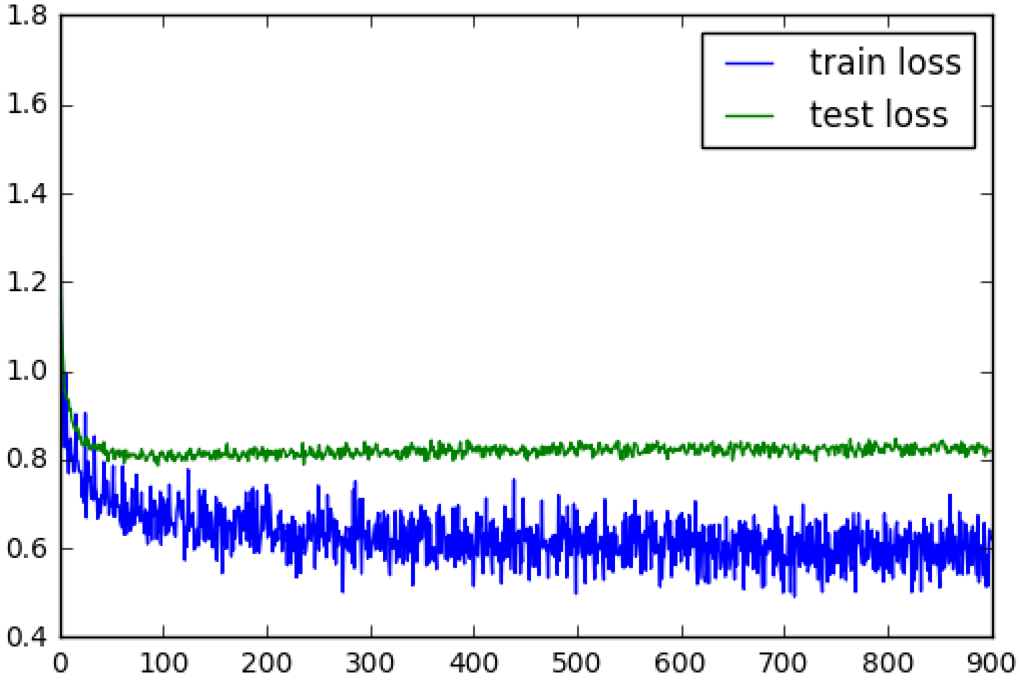
```
0.815308
[[843  80  20  31]
 [ 36 428  51  32]
 [  6  59 461  43]
 [ 11  38  26 131]]
```

In [42]:

```
tr_loss2 = []
tst_loss2 = []
tr_loss_overall = tr_loss + tr_loss2
tst_loss_overall = tst_loss + tst_loss2
x = range(len(tr_loss_overall))
plt.plot(x, tr_loss_overall, label='train loss')
plt.plot(x, tst_loss_overall, label='test loss')
plt.legend()
```

Out[42]:

```
<matplotlib.legend.Legend at 0x7f34feaf7e50>
```



In [62]:

```
# Save the model
saver = tf.train.Saver(write_version=1)
save_path = saver.save(sess, "context_models/neighbourhood_models/v6/mode
l.ckpt",)
print "Saved to:", save_path
```

```
WARNING:tensorflow:******************************************************
*
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:   `tf.train.Saver(write_version=tf.train.SaverDef.V2)
`
WARNING:tensorflow:now on by default.
WARNING:tensorflow:******************************************************
*
Saved to: context_models/neighbourhood_models/v6/model.ckpt
```

In [47]:

```
# Restore the model
saver = tf.train.Saver()
saver.restore(sess, "context_models/neighbourhood_models/v5/model.ckpt")
# v5 is actually slightly better than v6
```

In [48]:

```
# Evaluate f1
print sess.run(context_model.f1, feed_dict={
        context_model.probability_tensor:test['probabilities'],
        context_model.neighbourhood_tensor:test['weight_neighbourhoods'],
        context_model.label_tensor:test['labels'],
    })

# Show confusion matrix
print sess.run(context_model.confusion, feed_dict={
        context_model.probability_tensor:test['probabilities'],
        context_model.neighbourhood_tensor:test['weight_neighbourhoods'],
        context_model.label_tensor:test['labels'],
    })
```

```
0.816414
[[846  79  18  31]
 [ 35 432  49  31]
 [  7  62 453  47]
 [ 11  37  24 134]]
```

# Show Results

In [122]:

```
def show_results(patch, probabilities, neighbourhood, label, context_model):
    colours = { 0 : [255, 0, 0],
                1 : [0, 255, 0],
                2 : [0, 0, 255],
                3 : [255, 255, 0]}

    plt.figure(figsize=(10,8))
    plt.subplot(2, 2, 1)
    plt.imshow(patch)

    plt.subplot(2, 2, 2)
    pos = np.arange(4)+0.5
    barlist = plt.barh(pos, probabilities, align='center')
    barlist[0].set_color(np.array(colours[0]) / 255.0)
    barlist[1].set_color(np.array(colours[1]) / 255.0)
    barlist[2].set_color(np.array(colours[2]) / 255.0)
    barlist[3].set_color(np.array(colours[3]) / 255.0)
    plt.yticks(pos, categories)
    #plt.xlabel('Probability')
    plt.title('NEP Prediction')

    n = np.argmax(neighbourhood, axis=2)
    nrgb = np.zeros((5, 5, 3), dtype='uint8')
    for i in xrange(5):
        for j in xrange(5):
            nrgb[i][j] = colours[n[i][j]]
    plt.subplot(2, 2, 3)
    plt.imshow(nrgb)

    probabilities2 = sess.run(context_model.inference_predictions, feed_dict={
        context_model.probability_tensor:[probabilities],
        context_model.neighbourhood_tensor:[neighbourhood],
    })[0]
    plt.subplot(2, 2, 4)
    pos = np.arange(4)+0.5
    barlist = plt.barh(pos, probabilities2, align='center')
    barlist[0].set_color(np.array(colours[0]) / 255.0)
    barlist[1].set_color(np.array(colours[1]) / 255.0)
    barlist[2].set_color(np.array(colours[2]) / 255.0)
    barlist[3].set_color(np.array(colours[3]) / 255.0)
    plt.yticks(pos, categories)
    #plt.xlabel('Probability')
    plt.title('Context Prediction')

    print "Correct answer:", categories[np.argmax(label)]
```

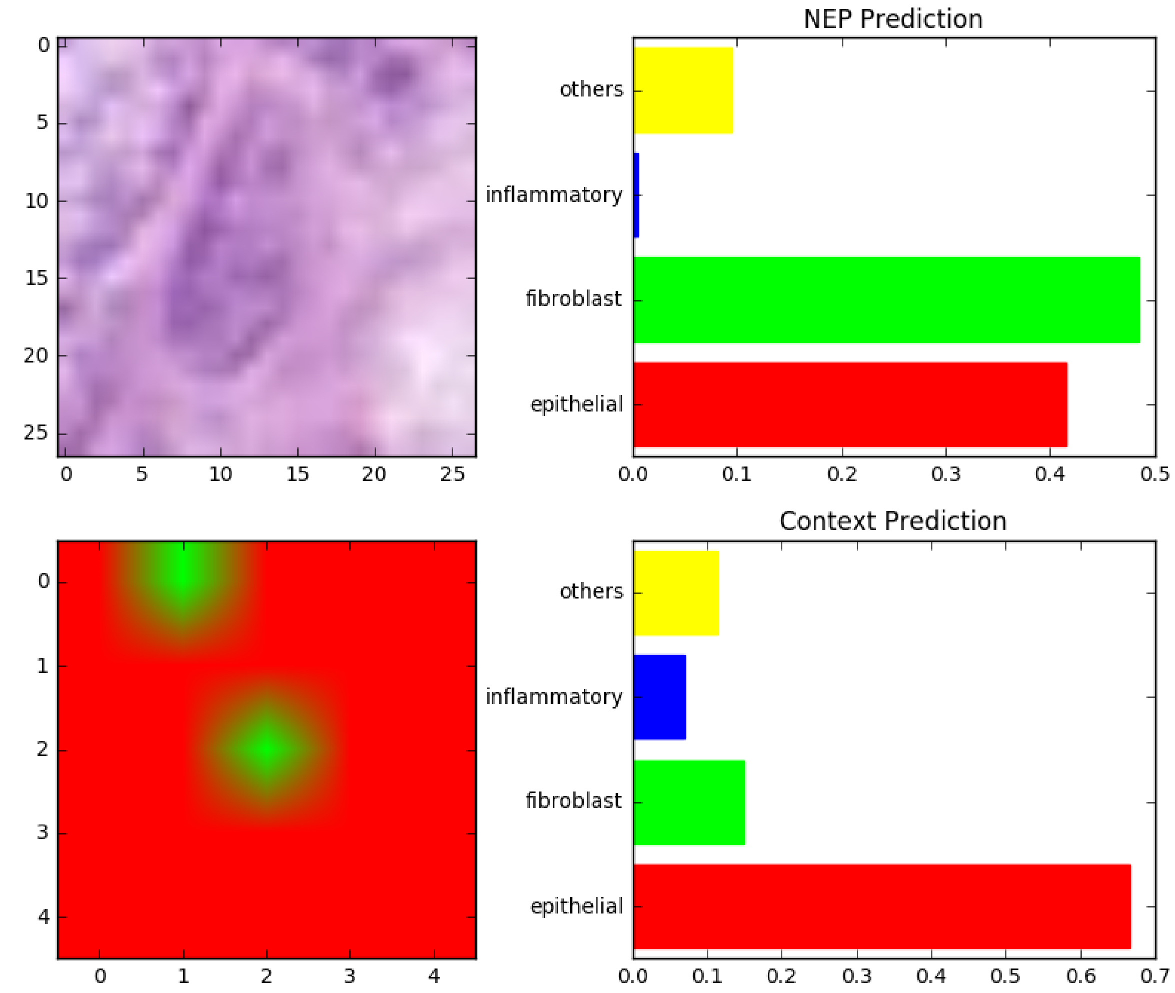
```
10
18
48
55
61
81
91
132
139
180
186
200
217
306
309
313
316
351
354
384
424
460
462
468
471
495
510
528
550
551
577
601
603
634
637
660
662
670
683
688
707
737
748
760
769
772
783
808
823
834
845
848
850
885
889
893
896
```

```
905
949
967
968
979
981
984
```

In [138]:

```
target = 984
show_results(test['patches'][target],
             test['probabilities'][target],
             test['weight_neighbourhoods'][target],
             test['labels'][target],
             context_model)
#print categories[np.argmax(test['labels'][target])]
```

```
Correct answer: epithelial
```


NEP Prediction
others
inflammatory
fibroblast
epithelial
Context Prediction
others
inflammatory
fibroblast
epithelial