In [11]:

```
import numpy as np
import imageio as imio
import matplotlib.pyplot as plt
%matplotlib inline
from skimage.filters import try_all_threshold as tat
from skimage.filters import threshold_otsu as otsu
from skimage.morphology import remove_small_objects as rso
from skimage.morphology import watershed
from skimage.feature import peak_local_max
from sklearn import preprocessing
from scipy import ndimage as ndi
import nibabel as nib
from scipy.stats import pearsonr
import os
from skimage.feature import match_template
import math
```

Functions

In [12]:

```
def plot_img(img):
    """ plots the raw weight matrices and the standardized wt. matrices """
    f = plt.figure()
    n_ = imio.imread('./neuron_{}.png'.format(img))[:,:,0]
    ax1 = plt.subplot(121)
    ax1.set_title('raw image')
    plt.imshow(n_, cmap='gray')
    n = n_.astype(float)
    n = (n - np.mean(n)) / np.std(n)
    ax2 = plt.subplot(122)
    ax2.set_title('stdized')
    plt.imshow(n, cmap='gray')
    return n
```

In [13]:

```
def spat_corr(sm_est, sm_gt, max_only=True, pos_th=0.6, neg_th=-0.6):
    """
    Determines the spatial correlations for all weight matrices againsts each ground truth compone
nt.

    Either picks the maximum correlation pair for each ground truth componen;
    or all matches above/below the given thresholds.

    """
#     sim_sm = list([sm_gt+i for i in os.listdir(sm_gt) if i.endswith('.nii')])
    sim_sm = sorted(list([sm_gt+i for i in os.listdir(sm_gt) if i.endswith('.nii')]), key= lambda x
: int(x.split('/')[-1].split('.')[0]))
    pos_sc = {}
    neg_sc = {}
    sm = {}
    spat_corrs = {}
    for n in range(1,129):
        wt = 'neuron_{}'.format(n)
        est_sm = sm_est.get(wt)
        n_sc = {}
        for nii in sim_sm:
            gt = nii.split('/')[-1].split('.')[0]
#             if int(gt) < 10: gt = '0{}'.format(gt)
            gt_sm_ = nib.load(nii)
            gt_sm = gt_sm_.get_fdata()
            sm[gt]=gt_sm
#             print(gt_sm.shape)
            sc = pearsonr(np.ravel(est_sm), np.ravel(gt_sm))[0]
            n_sc[gt] = sc
            if not max_only:
                if any((v>=pos_th or v<=neg_th) for k,v in n_sc.items()):
                    spat_corrs[wt+'_sm_{}'.format(gt)] = sc
```

```
        max_sc = max(n_sc, key=lambda key: n_sc[key])
        min_sc = min(n_sc, key=lambda key: n_sc[key])
        pos_sc[wt+'__sm_{}'.format(max_sc)] = n_sc[max_sc]
        neg_sc[wt+'__sm_{}'.format(min_sc)] = n_sc[min_sc]
    if not max_only: return spat_corrs, sm
    else: return pos_sc, neg_sc, sm
```

In [14]:

```
def spat_corr2(sm_est, sm_gt):

    sim_sm = list([sm_gt+i for i in os.listdir(sm_gt) if i.endswith('.nii')])
    pos_sc = {}
    neg_sc = {}
    sm = {}
    for nii in sim_sm:
        gt = nii.split('/')[-1]
        gt = gt.split('.')[0]
        gt_sm_ = nib.load(nii)
        gt_sm = gt_sm_.get_fdata()
        sm[gt]=gt_sm
        n_sc = {}
        for n in range(1,129):
            wt = 'neuron_{}'.format(n)
            est_sm = sm_est.get(wt)
            sc = pearsonr(np.ravel(gt_sm), np.ravel(est_sm))[0]
            n_sc[wt] = sc
        max_sc = max(n_sc, key=lambda key: n_sc[key])
        min_sc = min(n_sc, key=lambda key: n_sc[key])
        pos_sc['{}__sm_{}'.format(max_sc, gt)] = n_sc[max_sc]
        neg_sc['{}__sm_{}'.format(min_sc, gt)] = n_sc[min_sc]
    return pos_sc, neg_sc, sm
```

In [17]:

```
def plot_spat_corr(pos_spat_corr_dict, neg_spat_corr_dict, wt_dict, gt_dict, pos_th, neg_th):
    """ plots the spatially matched weight matrices and ground truth component maps """
    strong_corrs={}
    for k,v in pos_spat_corr_dict.items():
        if v>=pos_th:
            f = plt.figure()
            f.suptitle('{} corr={}'.format(k,v))
            ax1 = plt.subplot(121)
            ax1.set_title(k.split('__')[0])
            plt.imshow(wt_dict[k.split('__')[0]], cmap='gray')
            ax1 = plt.subplot(122)
            ax1.set_title(k.split('__')[1])
            plt.imshow(gt_dict[k.split('_')[-1]], cmap='gray')
            strong_corrs[k]=v
    for k,v in neg_spat_corr_dict.items():
        if v<=neg_th:
            f = plt.figure()
            f.suptitle('{} corr={}'.format(k,v))
            ax1 = plt.subplot(121)
            ax1.set_title(k.split('__')[0])
            plt.imshow(wt_dict[k.split('__')[0]], cmap='gray')
            ax1 = plt.subplot(122)
            ax1.set_title(k.split('__')[1])
            plt.imshow(gt_dict[k.split('_')[-1]], cmap='gray')
            strong_corrs[k]=v
    return strong_corrs
```

In [19]:

```
def plot_sm_fnc(spat_corr_dict, wt_dict):
    """
    For each weight matrix having at least one spatial correlation match
    greater than the designated thresholds - see spat_corr() -
    plots a bar graph showing component-wise spatial correlations.

    This is for identifying functional connectivity detections among the weight matrices

    """
    for wt in range(1,129):
```

```
        wt_sc = []
        for k,v in spat_corr_dict.items():
            if k.split('__')[0] == 'neuron_{}'.format(wt):# and (v>=pos_th or v<=neg_th):
                wt_sc.append([k.split('__')[1],v])#
#         print(wt_sc)
        wt_sc_sorted = sorted(wt_sc,key=lambda x: int(x[0].split('_')[1]))
#         print(wt_sc_sorted)
        if wt_sc:
            f = plt.figure(figsize=(12.8,4.8))
            f.suptitle('Wt matrix {}'.format(wt))
            plt.xticks(range(1,28))
            ax = plt.bar(list(int(wt_sc_sorted[i][0].split('_')[1]) for i in range(len(wt_sc_sorted
)),\
                         list(wt_sc_sorted[ii][1] for ii in range(len(wt_sc_sorted))))
#         plt.imshow(ax)
```

In [20]:

```
def max_corrs(pos_sm_corr, neg_sm_corr, wt_dict, gt_dict):
    for sm in range(1,28):
        f = plt.figure()
        f.suptitle('SM {}'.format(sm))
        sm_dict={}
        for k,v in pos_sm_corr.items():
            if k.split('_')[-1] == str(sm):
                sm_dict[k] = v
        if sm_dict:
            sm_max = max(sm_dict, key=lambda key: sm_dict[key])
            ax1 = plt.subplot(131)
            ax1.set_title('pos corr={0:.3f}'.format(sm_dict[sm_max]))
            plt.imshow(wt_dict[sm_max.split('__')[0]], cmap='gray')
        for k,v in neg_sm_corr.items():
            if k.split('_')[-1] == str(sm):
                sm_dict[k] = v
        if sm_dict:
            sm_min = min(sm_dict, key=lambda key: sm_dict[key])
            ax2 = plt.subplot(132)
            ax2.set_title('neg corr={0:.3f}'.format(sm_dict[sm_min]))
            plt.imshow(wt_dict[sm_min.split('__')[0]], cmap='gray')
        ax3 = plt.subplot(133)
        ax3.set_title('SM')
        plt.imshow(gt_dict[str(sm)], cmap='gray')
```

In [14]:

```
def preproc(img):
    """ not used; func. for basic image processing """
    th = otsu(img)
    img = img>th
    plt.subplot(131)
    plt.imshow(img, cmap='gray')
    smoothed = ndi.binary_fill_holes(img)
    smoothed = rso(smoothed, 25)
    plt.subplot(132)
    plt.imshow(smoothed, cmap='gray')
    cmpnts, nums = ndi.label(smoothed)
    plt.subplot(133)
    plt.imshow(cmpnts, cmap='gray')
    return cmpnts, nums
```

In [15]:

```
def show_objs(labels, nums):
    """ not used """
    for obj in range(1,nums+1):
        sbplt = 100+nums*10+obj
        plt.subplot(sbplt)
        plt.imshow(labels==obj, cmap='gray')
```

In [ ]:

```
## Analysis
```

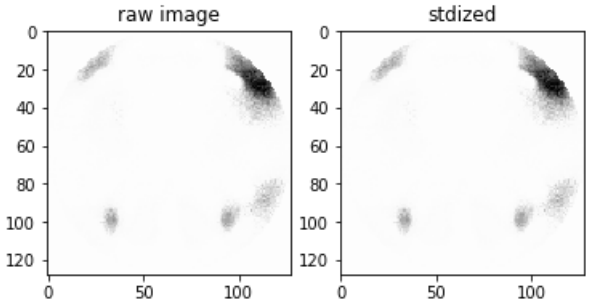
In [ ]:

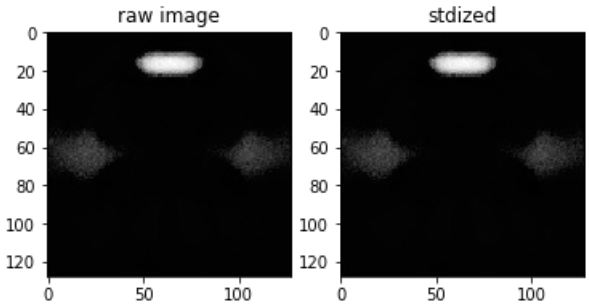
```
### Raw Weight Matrix Plots
```

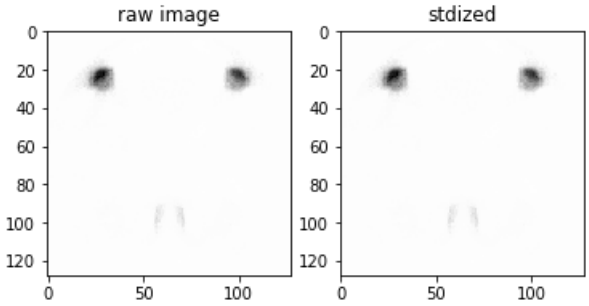
In [21]:

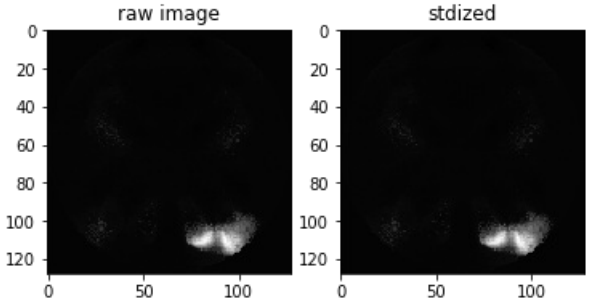
```
wts_stdz1 = {}
for img in range(1,129):
    n_stdz = plot_img(img)
    wts_stdz1['neuron_{}'.format(img)] = n_stdz
```

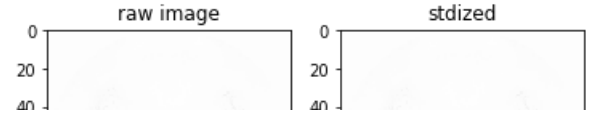
```
/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py:528: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

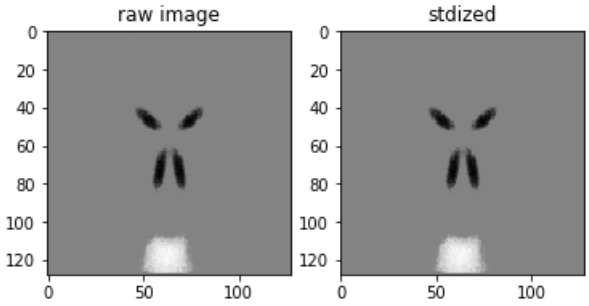












Order the weight matrix dictionary

In [1]:

```
# print(list(wts_stdz.keys()))
wts_stdz2 = sorted(wts_stdz1.items(), key=lambda kv: int(kv[0].split('_')[1]))
wts_stdz = {kv1[0]:kv1[1] for kv1 in wts_stdz2}
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-98b17b05243f> in <module>()
      1 # print(list(wts_stdz.keys()))
----> 2 wts_stdz2 = sorted(wts_stdz1.items(), key=lambda kv: int(kv[0].split('_')[1]))
      3 wts_stdz = {kv1[0]:kv1[1] for kv1 in wts_stdz2}

NameError: name 'wts_stdz1' is not defined
```

In [23]:

```
gt_cmpnts = '../../sim_SM/' # path to ground truth components
```

In [ ]:

```
### Calculate the spatial correlation
```

In [24]:

```
sc_pos, sc_neg, sm_cmpnts = spat_corr(wts_stdz, gt_cmpnts)
```

In [25]:

```
print(sm_cmpnts) # print to visually verify and ensure the key ordering is correct
```

```
{'1': array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]), '2': array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]), '3': array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]), '4': array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
```

```
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '18': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '19': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '20': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '21': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '22': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '23': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '24': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '25': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '26': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]), '27': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]])}
```

In [ ]:

```
#### Print the spatial correlations for visual check
```

In [26]:

```
for k,v in sc_pos.items():
    print(k,v)
```

```
neuron_1__sm_19 0.06498008112632055
neuron_2__sm_6 0.8598926414557981
```

```
neuron_99__sm_19 -0.7460417143829715
neuron_100__sm_4 -0.643584955359845
neuron_101__sm_12 -0.5843210334042065
neuron_102__sm_8 -0.688528702420908
neuron_103__sm_8 -0.7847931129910872
neuron_104__sm_16 -0.61274051395337
neuron_105__sm_24 -0.2205312489739852
neuron_106__sm_16 -0.7204450605028608
neuron_107__sm_7 -0.7457211377410266
neuron_108__sm_23 -0.597610825756708
neuron_109__sm_17 -0.7964110876739017
neuron_110__sm_23 -0.838609288318103
neuron_111__sm_14 -0.055853968152359136
neuron_112__sm_2 -0.8275091512999447
neuron_113__sm_17 -0.060254221175940016
neuron_114__sm_26 -0.8910147948493624
neuron_115__sm_24 -0.6760439554375975
neuron_116__sm_9 -0.9032984535753004
neuron_117__sm_13 -0.6837561204321733
neuron_118__sm_7 -0.8936763746857302
neuron_119__sm_16 -0.6574744951992455
neuron_120__sm_14 -0.557078238966414
neuron_121__sm_14 -0.06922938224326948
neuron_122__sm_14 -0.7540937259548318
neuron_123__sm_16 -0.06656730961499144
neuron_124__sm_16 -0.041579974872025235
neuron_125__sm_16 -0.549888378450922
neuron_126__sm_12 -0.6414333564248381
neuron_127__sm_16 -0.7469475994227327
neuron_128__sm_12 -0.5068038710761519
```

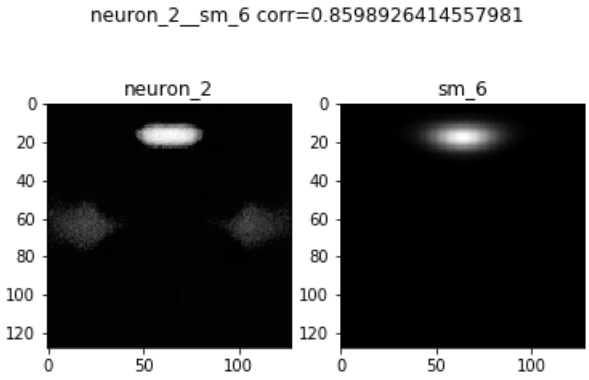
In [ ]:

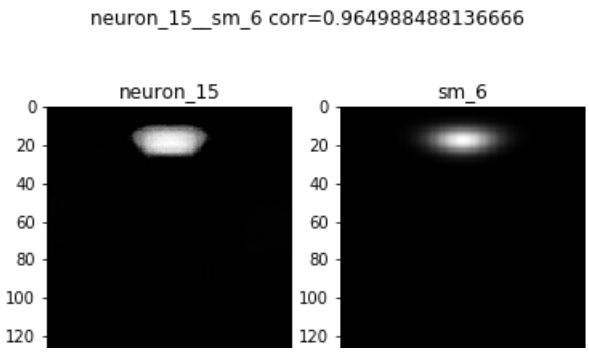
```
### Plot the spatial correlations matches (max for each weight matrix)
```

In [30]:

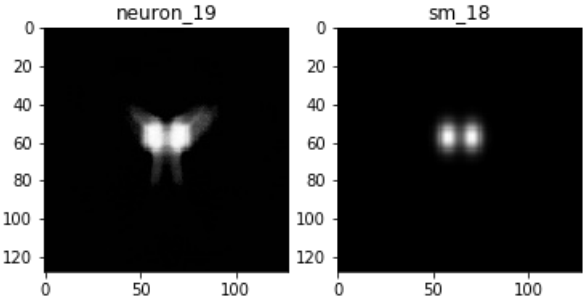
```
spat_corrs = plot_spat_corr(sc_pos, sc_neg, wts_stdz, sm_cmpnts, 0.6, -0.6)
```

```
/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py:528: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

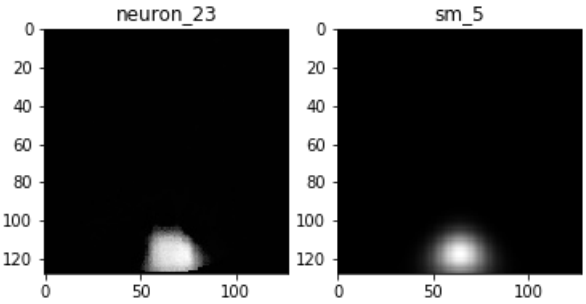




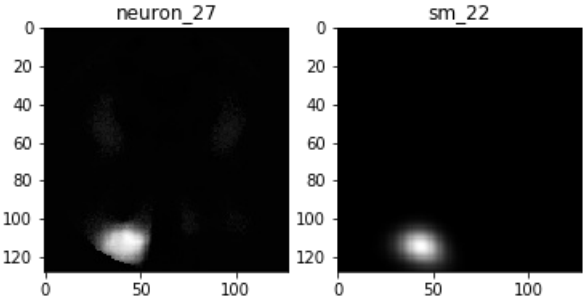
neuron_19__sm_18 corr=0.9379361994414107



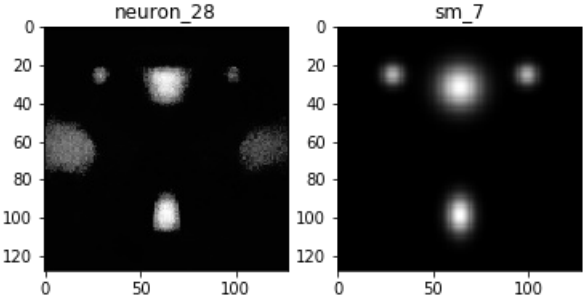
neuron_23__sm_5 corr=0.9241423396778127



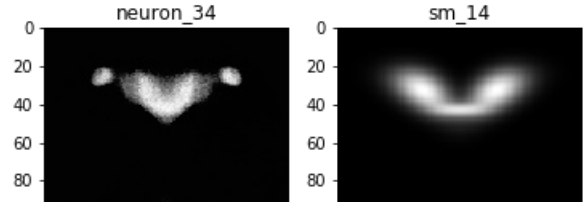
neuron_27__sm_22 corr=0.9315828911375861



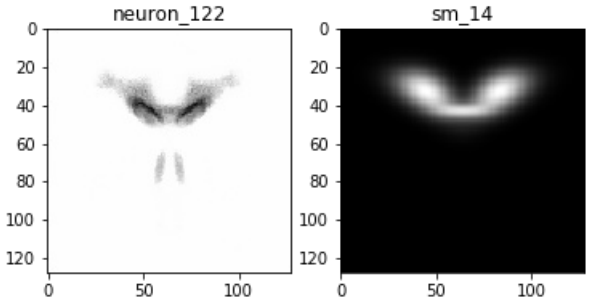
neuron_28__sm_7 corr=0.7574517598824874



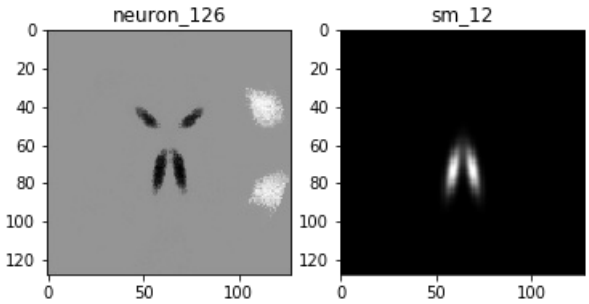
neuron_34__sm_14 corr=0.6867895643020699



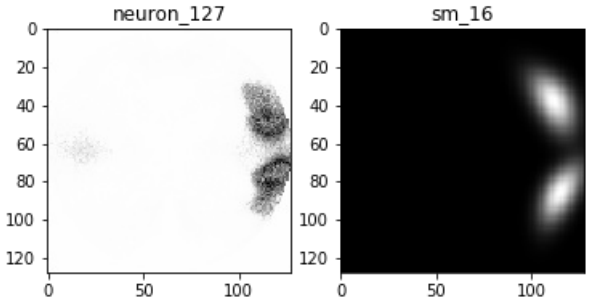
neuron_122__sm_14 corr=-0.7540937259548318



neuron_126__sm_12 corr=-0.6414333564248381



neuron_127__sm_16 corr=-0.7469475994227327



In [ ]:

```
### Calculate and plot the spatial correlations (max for each g.t. component)
```

In [31]:

```
sc_pos2, sc_neg2, _ = spat_corr2(wts_stdz, gt_cmpnts)
```

In [32]:

```
for k,v in sc_pos2.items():
    print(k,v)
```

```
neuron_70__sm_1 0.46647175964757115
neuron_48__sm_10 0.9739256377241232
neuron_14__sm_11 0.5448878695392745
neuron_14__sm_12 0.5837966719458684
neuron_77__sm_13 0.29328949431891377
neuron_34__sm_14 0.6867895643020699
neuron_111__sm_15 0.6152492108158875
neuron_126__sm_16 0.5145105074696585
neuron_54__sm_17 0.9470167340201475
```

```
neuron_19__sm_18 0.9379361994414107
neuron_44__sm_19 0.5103727150362471
neuron_104__sm_2 0.0840430639804995
neuron_97__sm_20 0.5228024201474543
neuron_121__sm_21 0.81014007294113
neuron_27__sm_22 0.9315828911375861
neuron_50__sm_23 0.9733055949746792
neuron_9__sm_24 0.3202036402579397
neuron_41__sm_25 0.32514328793673625
neuron_113__sm_26 0.7530099474518054
neuron_124__sm_27 0.9496851302441098
neuron_27__sm_3 0.18373281476756612
neuron_4__sm_4 0.4768652285809789
neuron_23__sm_5 0.9241423396778127
neuron_15__sm_6 0.964988488136666
neuron_35__sm_7 0.9110710816063504
neuron_45__sm_8 0.9753544236245922
neuron_15__sm_9 0.06666965546098608
```

In [33]:

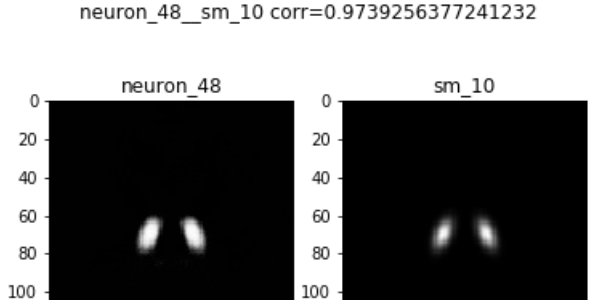
```
for k,v in sc_neg2.items():
    print(k,v)
```

```
neuron_51__sm_1 -0.9520506120687706
neuron_91__sm_10 -0.985503678472384
neuron_11__sm_11 -0.5476535641495032
neuron_11__sm_12 -0.7249076535606194
neuron_26__sm_13 -0.9465524484483397
neuron_13__sm_14 -0.833104630736732
neuron_44__sm_15 -0.7516096930400037
neuron_17__sm_16 -0.8288980135293368
neuron_66__sm_17 -0.880364182054777
neuron_128__sm_18 -0.09631588369632121
neuron_29__sm_19 -0.7544174993929196
neuron_88__sm_2 -0.9359822914066607
neuron_18__sm_20 -0.7964327561808563
neuron_30__sm_21 -0.8814243575418048
neuron_31__sm_22 -0.34865416150551964
neuron_110__sm_23 -0.838609288318103
neuron_58__sm_24 -0.8306547476742862
neuron_24__sm_25 -0.8709505222933924
neuron_12__sm_26 -0.9354790927534705
neuron_36__sm_27 -0.972306388812063
neuron_5__sm_3 -0.9486357039585148
neuron_63__sm_4 -0.9244934497093861
neuron_95__sm_5 -0.8754652477562993
neuron_51__sm_6 -0.19986939146837587
neuron_118__sm_7 -0.8936763746857302
neuron_103__sm_8 -0.7847931129910872
neuron_53__sm_9 -0.9099764677612737
```

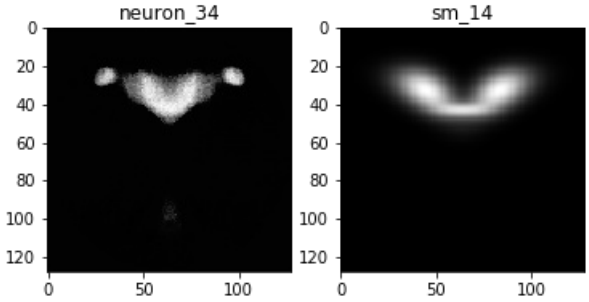
In [34]:

```
spat_corrs2 = plot_spat_corr(sc_pos2, sc_neg2, wts_stdz, sm_cmpnts, 0.6, -0.6)
```

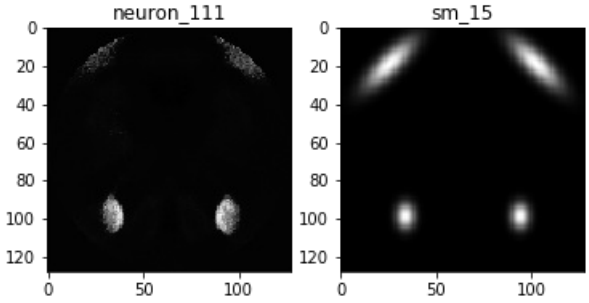
```
/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py:528: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



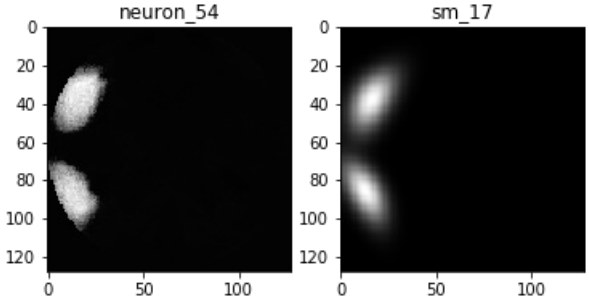
neuron_34__sm_14 corr=0.6867895643020699



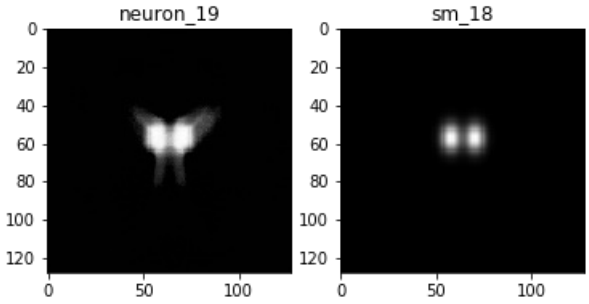
neuron_111__sm_15 corr=0.6152492108158875



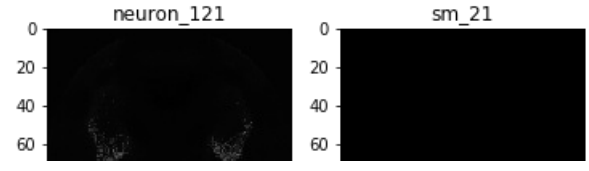
neuron_54__sm_17 corr=0.9470167340201475

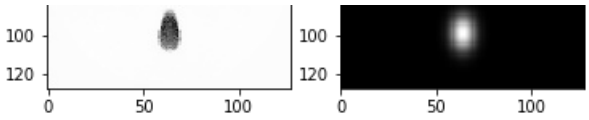


neuron_19__sm_18 corr=0.9379361994414107

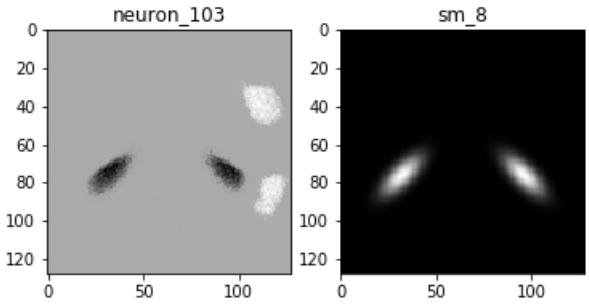


neuron_121__sm_21 corr=0.81014007294113

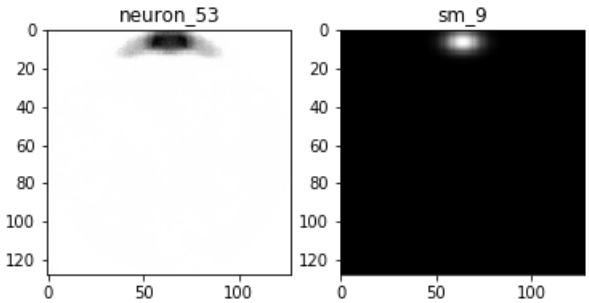




neuron_103__sm_8 corr=-0.7847931129910872



neuron_53__sm_9 corr=-0.9099764677612737



In [ ]:

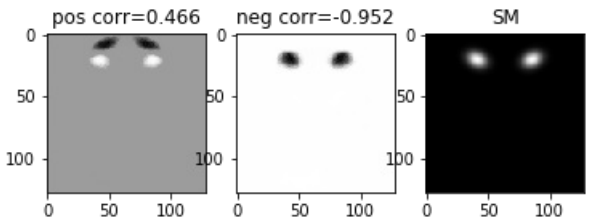
```
# Plot the max positive and negative correlations for each ground truth
```

In [35]:

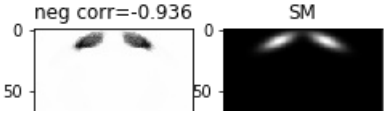
```
max_corrs(sc_pos, sc_neg, wts_stdz, sm_cmpnts)
```

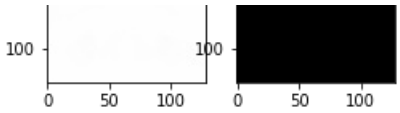
```
/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py:528: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

SM 1

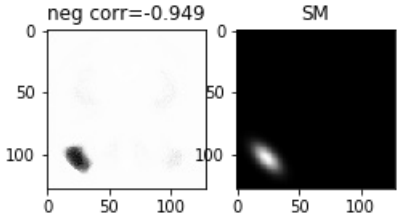


SM 2

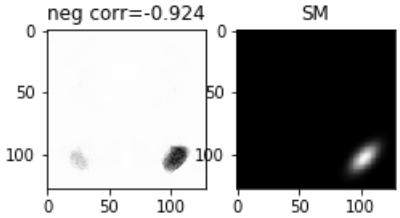




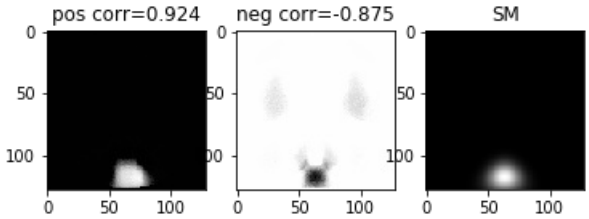
SM 3



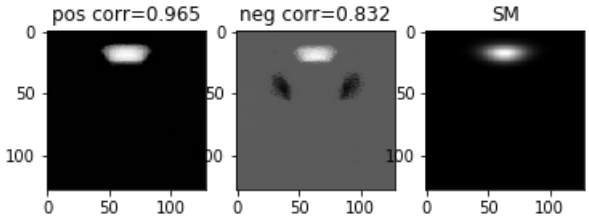
SM 4



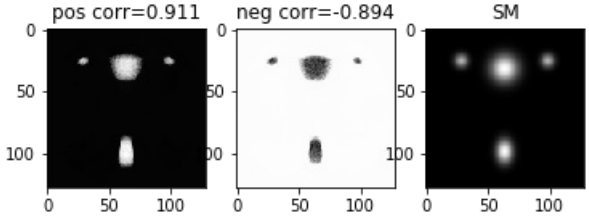
SM 5

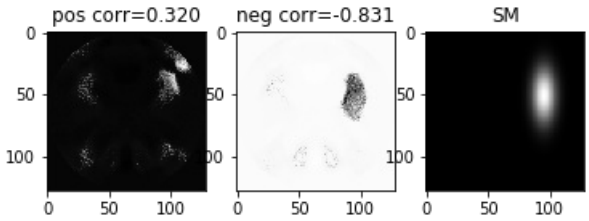


SM 6

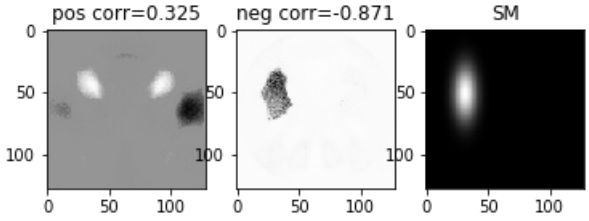


SM 7

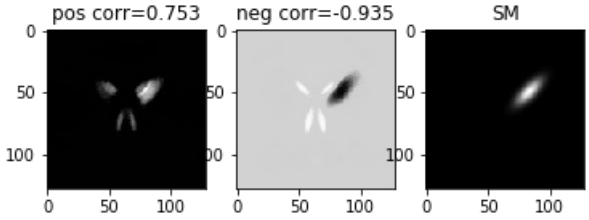




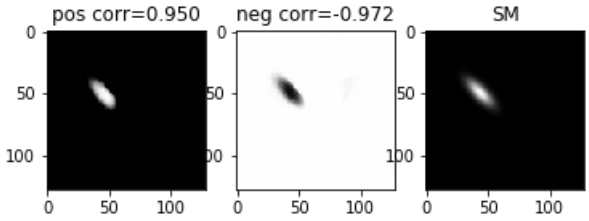
SM 25



SM 26



SM 27



In [ ]:

```
#### 26 of 27 (96.3%) of the ground truth components were captured by the RBM
##### the 27th (SM 11) visually has matches, but was not captured by pearson corr.
```

In [ ]:

```
### Calculate the spatial correlations above a given threshold
```

In [103]:

```
scorrs, _ = spat_corr(wts_stdz, gt_cmpnts, max_only=False)
```

In [105]:

```
scorrs
```

Out[105]: