Unsupervised Learning 2 - Other techniques (on Tileset7) - Aug 2017

Created: 16 Aug 2018
Last update: 16 Aug 2018

Use some more unsupervised techniques learned from DataCamp

This continues the work from 'realxtals1-unsupervised1.ipynb'. Some of the functions of that notebook have now been moved into imgutils plus some other extensions in imgutils visualization (mostly adding the 'large heatmap' capability and extra annotations)

About the data: The data used here has been prepared in my prior notebooks. It's a bunch of images (from a larger 'tile set') sliced up in sub-images and image statistics applied on each sub-image.

---

1. Imports

In [337]:
```
# this will remove warnings messages
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

# import
from sklearn import cluster
from sklearn.preprocessing import LabelEncoder

import imgutils
```

In [338]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)
```

Out[338]:
```
<module 'imgutils' from 'C:\\JADS\\SW\\Grad Proj\\realxtals1\\sources\\imgutils.py'>
```

---

2. Import Crystal Image Data & Statistics

The data was labeled and exported to csv in the notebook realxtals1_dataeng1.ipynb

About the data:

The CSV contains the image files, slice information (sub-images) and associated statistics, which are the features for which a classifier needs to be found.

The goal is to find the clustering in feature-space and use those to categorize the images. For this particular dataset, a single statistics could be used to label into three classes:

A = subimage contains no crystal,
B = part of subimage contains crystal,
C = (most of) subimage contains crystal

But the labels have been added here for analyses, eventually the data will be unlabelled.

Import data:

In [312]:
```
df = pd.read_csv('../data/Crystals_Apr_12/Tileset7-2.csv', sep=';')
df.head(3)

imgnames = df3['filename'].unique()
```

---

3. Re-do some of the clustering from previous notebook

(so we have some comparison material)

First vectorize the data:

In [82]:
```
# convert into X Y vectors:
feature_cols = ['|img_std|', '|img_std2|', '|img_mean|','|img_skewness|', '|img_kurtosis|','|img_mode|']
X = df.loc[:,feature_cols]
```

k-means:

In [297]:
```
number_of_clusters = 3
```

In [306]:
```
k_means = cluster.KMeans(algorithm='auto', n_clusters=3, n_init=10, init='k-means++')
k_means.fit(X)
k_means_pred= k_means.labels_
```

Hierarchical clustering

In [308]:
```
from sklearn.cluster import AgglomerativeClustering

# Affinity = {“euclidean”, “l1”, “l2”, “manhattan”, “cosine”}
# Linkage = {“ward”, “complete”, “average”}

Hclustering = AgglomerativeClustering(n_clusters=3, affinity='cosine', linkage='complete')
Hclustering.fit(X)
hierarch_pred = Hclustering.labels_
```

Spectral and DBScan:

In [343]:
```
spectral = cluster.SpectralClustering(n_clusters=number_of_clusters,eigen_solver='arpack',affinity="nearest_neighbors")
spectral.fit(X)
spectral_pred = spectral.labels_

dbscan = cluster.DBSCAN(eps=0.5, metric='euclidean', min_samples=8)
dbscan.fit(X)
dbscan_pred = dbscan.labels_
```

Also get the PCA transformed data and k-means and hierach with PCA

In [349]:
```
from sklearn import decomposition

fieldnames = ['pca_1','pca_2','pca_3', 'pca_4', 'pca_5']

n_comp = 5;

pca = decomposition.TruncatedSVD(n_components=n_comp)
X_fit = pca.fit_transform(X)

# convert into X Y vectors:
df_pca = pd.DataFrame(X_fit[:,0:n_comp], columns=fieldnames[:n_comp])
X_pca = df_pca.loc[:,fieldnames[:n_comp]]
```

In [360]:
```
k_means_pca = cluster.KMeans(algorithm='auto', n_clusters=3, n_init=10, init='k-means++')
k_means_pca.fit(X)
k_means_pca_pred = k_means_pca.labels_
```

In [361]:
```
Hclustering_pca = AgglomerativeClustering(n_clusters=3, affinity='cosine', linkage='complete')
Hclustering_pca.fit(X_pca)
hierarch_pca_pred = Hclustering_pca.labels_
```

---

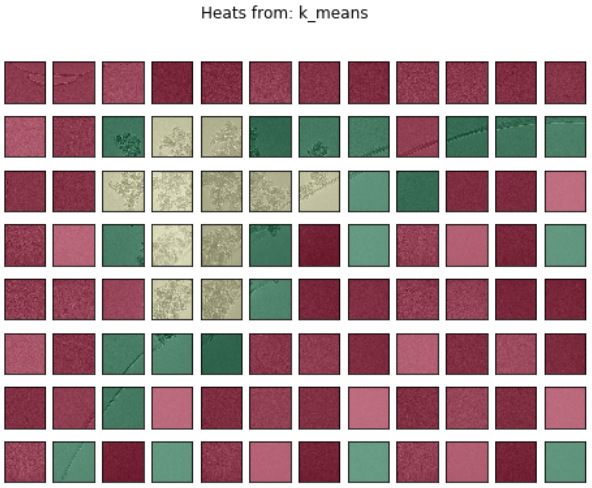
4. Visualize k-means and hierarchical clustering and assess scores

(The large heat map is now part of imgutils)

In [350]:
```
# Add the unsupervised clustering results to the dataframe
df3 = df
df3['k_means'] = k_means_pred
df3['hierarch'] = hierarch_pred

figsize=(8,6)
```

In [351]:
```
# show heatmaps:
imgutils.show_large_heatmap(df3, 'k_means', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```



In [352]:
```
imgutils.show_large_heatmap(df3, 'hierarch', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```

The baseline score of these two with manual counting

(see previous notebook (unsupervised1). The idea is to count true positives and false positives on the important categories)

In [345]:
```
def count_imgs_per_class(df_imgstats, classcolumn):
    return df_imgstats[classcolumn].value_counts()

def print_scores(methodname, class_count_tuples):
    """
    the class tuple has form (classname, n_true_pos, n_false_pos, n_real_pos)
    """
    print("")
    print("{:<20}|{:^12}|{:^12}|".format(methodname.upper(), "True Pos", "False Pos"))
    print("-"*(20+12+12+3))

    def print_score_line(class_name, TPR, FDR):
        print("{:<20}|{:^12.2%}|{:^12.2%}| ".format(class_name, TPR, FDR ))

    for (class_name, n_true_pos, n_false_pos, n_real_pos) in class_count_tuples:
        TPR = n_true_pos/n_real_pos
        FDR = n_false_pos/(n_true_pos + n_false_pos)
        print_score_line(class_name, TPR, FDR)

    print("-"*(20+12+12+3))
```

In [346]:
```
print_scores('Manual (using STD)', [('Full Crystal', 11, 2, 11),  ('Partial Crystal', 6, 4, 8) ])
print_scores('Hierarchical', [('Full Crystal', 11, 1, 11), ('Partial Crystal', 7, 12, 8) ])
print_scores('K-means', [('Full Crystal', 10, 1, 11), ('Partial Crystal', 7, 13, 8) ])
```

```
MANUAL (USING STD)  |  True Pos  | False Pos  |
-----------------------------------------------
Full Crystal        |  100.00%   |   15.38%   |
Partial Crystal     |   75.00%   |   40.00%   |
-----------------------------------------------

HIERARCHICAL        |  True Pos  | False Pos  |
-----------------------------------------------
Full Crystal        |  100.00%   |   8.33%    |
Partial Crystal     |   87.50%   |   63.16%   |
-----------------------------------------------

K-MEANS             |  True Pos  | False Pos  |
-----------------------------------------------
Full Crystal        |   90.91%   |   9.09%    |
Partial Crystal     |   87.50%   |   65.00%   |
-----------------------------------------------
```

REMARKS:

- some of the false positives in category 'Partial' are 'Full Ones' and some false positives in 'Full' are partial ones. So this score is a bit to strict, but accounting for this would require more complex scoring (or a full confusion matrix, where you still need to remark that some confusion is not so critical)
- running the algorithms gives some variation, so these scores may deviate a bit (it is manually counted)

---

5. 'Unsupervise scoring' based on cluster 'shape' (instead of using ground truth labels)

The silhouette-score assesses 'cluster consistency' in a single number, and is part of sklearn package see https://en.wikipedia.org/wiki/Silhouette_(clustering (https://en.wikipedia.org/wiki/Silhouette_(clustering))

In [353]:
```
from sklearn.metrics import silhouette_score
```

In [354]:
```
def print_score(name, data, labels):
    print("%s: %f" % (name,silhouette_score(data, labels)))
```

Scores for k-means, spectral, dbscan and hierarchical

In [358]:
```
print_score('k-means', X, k_means_pred)
print_score('spectral', X, spectral_pred)
print_score('dbscan', X, dbscan_pred)
print_score('hierarchical', X, hierarch_pred)
```

```
k-means: 0.525192
spectral: 0.174689
dbscan: 0.355807
hierarchical: 0.486214
```

And the pca variants

In [362]:
```
print_score('k-means PCA', X_pca, k_means_pca_pred)
print_score('hierarchical PCA', X_pca, hierarch_pca_pred)
```

```
k-means PCA: 0.525208
hierarchical PCA: 0.486230
```

the intrinsic scoring slighlty prefers k-means

---

6. Check some other interesting properties (from DataCamp)

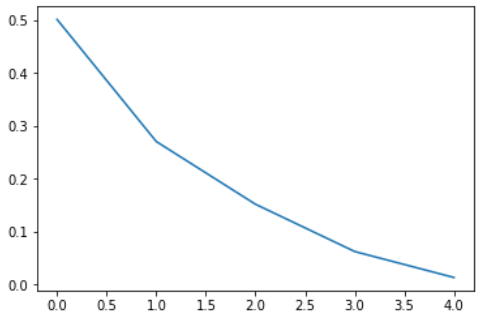
Examine PCA components importance, with the 'variance explained' of the PCA

In [371]:
```
var_ex = pca.explained_variance_ratio_
print(var_ex)
```

```
[0.50195426 0.27067401 0.1518714  0.06234576 0.01312667]
```

In [372]:
```
plt.plot(var_ex)
```

Out[372]:
```
[<matplotlib.lines.Line2D at 0x1dd5ed30>]
```



Hard to point out 'elbow', but from the values indeed the first two or three are significant

And look at the correlation between statistics

In [373]:
```
from scipy.stats import pearsonr

print(feature_cols)
```

```
['|img_std|', '|img_std2|', '|img_mean|', '|img_skewness|', '|img_kurtosis|', '|img_mode|']
```

In [374]:
```
print("std - std2:", pearsonr(df['|img_std|'], df['|img_std2|']))
print("std - mean:", pearsonr(df['|img_std|'], df['|img_mean|']))
print("mean - kurtosis:", pearsonr(df['|img_mean|'], df['|img_kurtosis|']))
print("mean - skewness:", pearsonr(df['|img_mean|'], df['|img_skewness|']))
print("kurtosis - skewness:", pearsonr(df['|img_kurtosis|'], df['|img_skewness|']))
print("mean - mode:", pearsonr(df['|img_mean|'], df['|img_mode|']))
```

```
std - std2: (0.9984151451987054, 2.7965952882845724e-119)
std - mean: (-0.497753236525996, 2.472851861616533e-07)
mean - kurtosis: (0.11489907756487519, 0.2649679416831979)
mean - skewness: (0.023172757326829583, 0.8226782343049519)
kurtosis - skewness: (-0.6036063134052272, 7.541338508275421e-11)
mean - mode: (-0.1027398641558728, 0.3192047878221636)
```

(the first value is the correction coefficient, second a p-value; see https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.pearsonr.html (https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.pearsonr.html))

As expected, std and std2 are highly correlated. Interesting/puzzling that they can be combined to identify clusters

Correlation between PCA components?

In [369]:
```
print("pca_1 - pca_2:", pearsonr(df_pca['pca_1'], df_pca['pca_2']))
print("pca_1 - pca_3:", pearsonr(df_pca['pca_1'], df_pca['pca_3']))
print("pca_2 - pca_3:", pearsonr(df_pca['pca_2'], df_pca['pca_3']))
```

```
pca_1 - pca_2: (5.918314603462634e-17, 1.0)
pca_1 - pca_3: (-3.386152714572967e-17, 1.0)
pca_2 - pca_3: (2.075046571669496e-16, 1.0)
```

Indeed, as expected, pca components are uncorrelated.

7.NMF Similarity (Non-Negative Matrix Factorization)

The idea is as follows:

- standardize the data in such a way that is has no zero values
- apply NMF
- normalize each sample (so their feature vector has lenght 1),
- pick one sample as the reference
- perform a dot product of all samples with the reference samples;
- if feature vectors are very similar, the dot product will be close to one

If we pick a sub image with clear x-crystals as the refernece, see how well this dot product works for a heat map

More info on NMF: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html) (and on DataCamp)

In [375]:
```
import sklearn.preprocessing as skpreproc
from sklearn.decomposition import NMF

# assure the all data is non negative and in sane rane
X_scaled = skpreproc.minmax_scale(X)

nmf = NMF()
X_nmf = nmf.fit_transform(X_scaled)

# for feature comparison, each feature vector should be normalized (i.e. per sample)
X_nmf_norm = skpreproc.normalize(X_nmf)

df_nmf = pd.DataFrame(X_nmf_norm)
df_nmf.head(5)
```

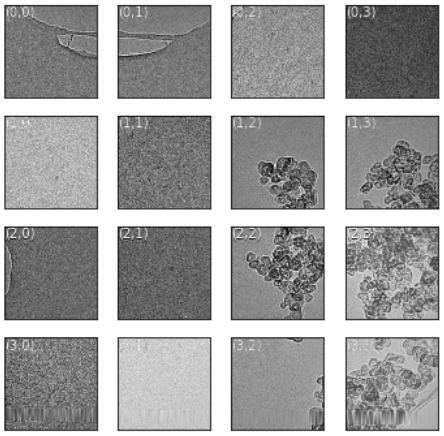
Out[375]:

	0	1	2	3	4	5
0	0.757494	0.208443	0.141452	0.087992	0.215789	0.555372
1	0.625969	0.263954	0.383905	0.201435	0.382042	0.452300
2	0.853773	0.109551	0.387315	0.059117	0.103158	0.308093
3	0.851769	0.064229	0.461585	0.000000	0.042644	0.235554
4	0.744275	0.131432	0.107639	0.146227	0.029009	0.628466

Now let's select a reference tile from the first image

In [377]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)

img1, dummy = imgutils.getimgslices_fromdf(df, imgnames[0])
imgutils.showimgs(img1, tile_labels=True, fig_size=(6,6))
```



Let's use tile (2,3) as the reference image 'with crystal'; need to determine it's row number...

In [391]:
```
print(df[(df['filename']==imgnames[0]) & (df['s_y']==2) &  (df['s_x']==3)].iloc[:,2:10])
```

```
    s_y  s_x  n_y  n_x     alias    img_mean      img_std  img_kurtosis
11    2    3    4    4  img0_2-3  8016.20884  1879.269345     -1.035852
```

Ah, that is why I introduced alias, so I can just use

In [393]:
```
df[df['alias']=="img0_2-3"]
```

Out[393]:

	Unnamed: 0	filename	s_y	s_x	n_y	n_x	alias	img_mean	img_std	img_kurtosis	...	img_std2	|img_
11	11	..\data\Crystals_Apr_12\Tileset7\Tile_001-001-...	2	3	4	4	img0_2-3	8016.20884	1879.269345	-1.035852	...	0.234434	-3.619

1 rows × 22 columns

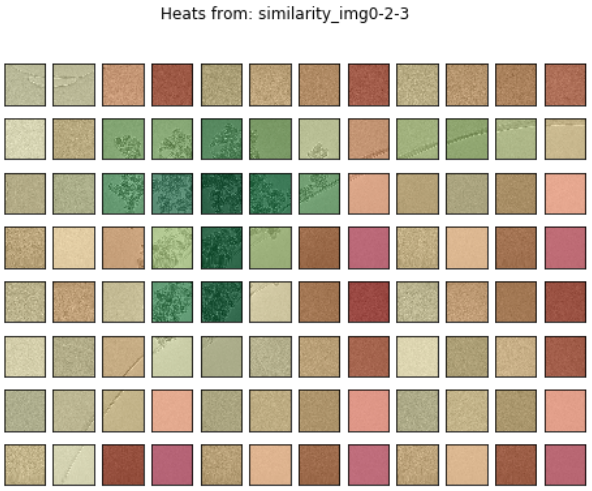
Anyway, it's row 11

In [395]:
```
ref_vect = X_nmf_norm[11]
similarities = df_nmf.dot(ref_vect)

# check if indeed similarity of row 11 is 1
print(similarities[10:13])
```

```
10    0.845384
11    1.000000
12    0.374758
dtype: float64
```

In [399]:
```
# assign this to the dataframe and then use this as heats
df3['similarity_img0-2-3'] = similarities
imgutils.show_large_heatmap(df3, 'similarity_img0-2-3', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```

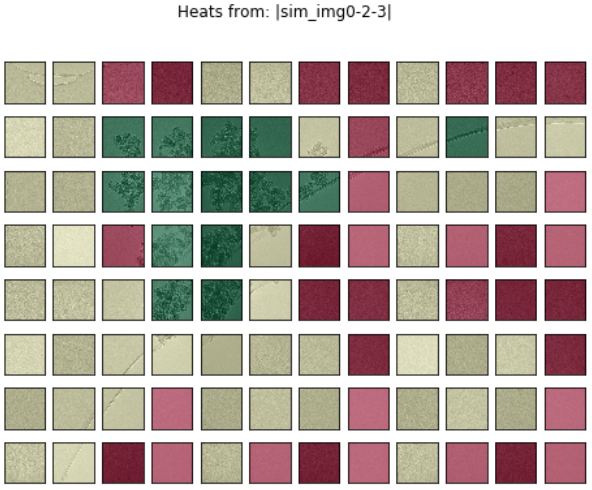


Hmmm, hard to assess, let's map to integer

In [401]:
```
df3['|sim_img0-2-3|'] = df3['similarity_img0-2-3'].map(lambda x: int(x * 3 - 0.0001))
df3['|sim_img0-2-3|'].value_counts()
```

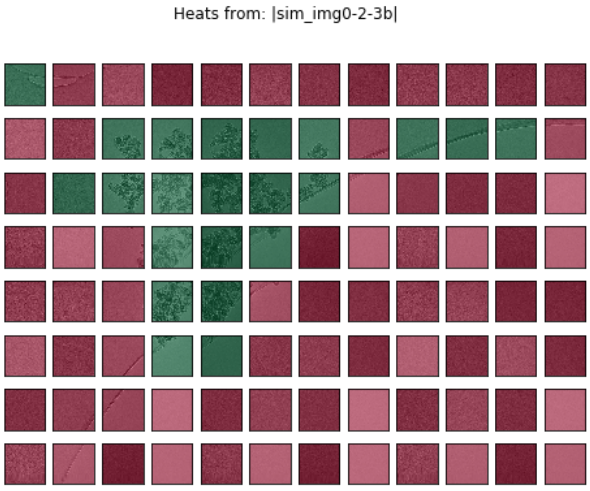
Out[401]:
```
1    48
0    34
2    14
Name: |sim_img0-2-3|, dtype: int64
```

In [402]:
```
imgutils.show_large_heatmap(df3, '|sim_img0-2-3|', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```



Hmmm, color coding can be misleading. What if I just define two clusters?

In [403]:
```
df3['|sim_img0-2-3b|'] = df3['similarity_img0-2-3'].map(lambda x: int(x * 2 - 0.0001))
df3['|sim_img0-2-3b|'].value_counts()
imgutils.show_large_heatmap(df3, '|sim_img0-2-3b|', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```



Ok. Let's also try without NMF decomposition (just similarity via dot product of original feature vectors). But first assess scores.

Clustering score

In [404]:
```
# we can assess the clustering in either 'original space' or in NMF transformed space:
print_score('NMF (original)', X, df3['|sim_img0-2-3|'] )
print_score('NMF Transformed', X_nmf, df3['|sim_img0-2-3|'] )
```

```
NMF (original): 0.231367
NMF Transformed: 0.200687
```

Not very high...

---

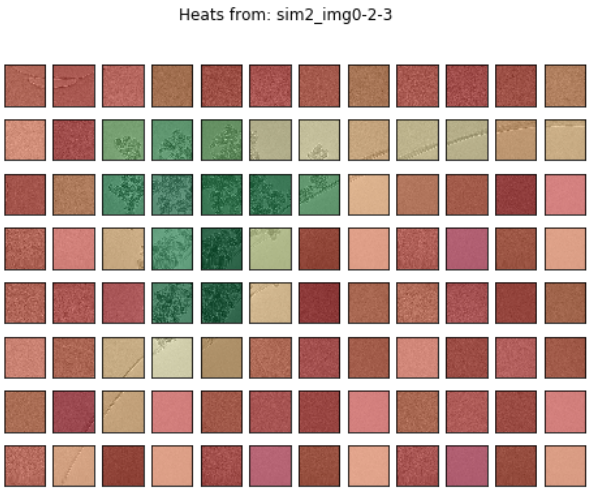
8. Similarity without any transformation

In [407]:
```
X_norm = skpreproc.normalize(X) # the orignal features, but normalized per sample
ref_vect2 = X_norm[11]
df3['sim2_img0-2-3'] = X_norm.dot(ref_vect2)

# do a quick check; element 11 should have value 1
df3['sim2_img0-2-3'].iloc[9:13]
```

Out[407]:
```
9    -0.284271
10    0.851739
11    1.000000
12   -0.473861
Name: sim2_img0-2-3, dtype: float64
```

In [408]:
```
imgutils.show_large_heatmap(df3, 'sim2_img0-2-3', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```

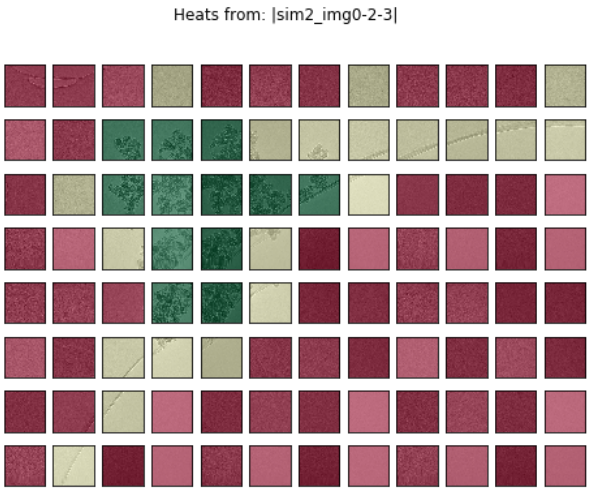


This looks actually pretty good. For a heatmap the gradual scale is nice, but for classification we need to reduce this to e.g. 3 clusters to compare it's intrinsic score to the other approaches

In [413]:
```
# as we observed negatives, we are going to make again 3 categories
# but note that range is now -1 to +1,
df3['|sim2_img0-2-3|'] = df3['sim2_img0-2-3'].map(lambda x: int( (x+1) * 3 / 2 - 0.0001))
df3['|sim2_img0-2-3|'].value_counts()
```

Out[413]:
```
0    64
1    20
2    12
Name: |sim2_img0-2-3|, dtype: int64
```

In [410]:
```
imgutils.show_large_heatmap(df3, '|sim2_img0-2-3|', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```



Scoring with silhouette

In [412]:
```
# assess the clustering in 'original space' and 'rescaled space':
print_score('Similarity (original)', X, df3['|sim2_img0-2-3|'] )
print_score('Similarity (scaled)', X_norm, df3['|sim2_img0-2-3|'] )
```

```
Similarity (original): 0.425053
Similarity (scaled): 0.380463
```

A better way to score this would be using the assessment from the heatmap, which involves counting :-(

In [277]:
```
print_scores('Similarity', [('Full Crystal', 11, 1, 11),  ('Partial Crystal', 7, 13, 8) ])
```

```
SIMILARITY          |  True Pos  | False Pos  |
-------------------------------------------------
Full Crystal        |  100.00%   |   8.33%    |
Partial Crystal     |   87.50%   |   65.00%   |
-------------------------------------------------
```

Again, the false positive of the full-crystal is debatable.

In general, I should maybe even aim for a more binary classification with or without)

---

9. Similarity based on PCA components

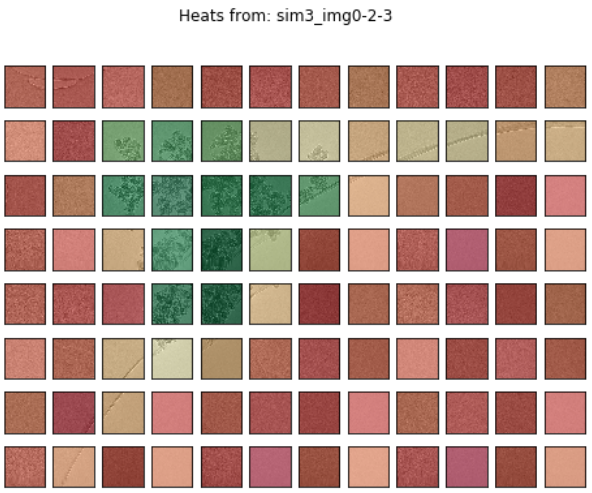
In [414]:
```
# normalize the PCA features per sample so we can compare the feature vectors
X_pca_norm = skpreproc.normalize(X_pca)

# compare via dot product with the reference image
ref_vect3 = X_pca_norm[11]
df3['sim3_img0-2-3'] = X_pca_norm.dot(ref_vect3)

# do a quick check; element 11 should have value 1
df3['sim3_img0-2-3'].iloc[9:13]
```

Out[414]:
```
9    -0.284296
10    0.851738
11    1.000000
12   -0.473850
Name: sim3_img0-2-3, dtype: float64
```

In [415]:
```
imgutils.show_large_heatmap(df3, 'sim3_img0-2-3', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```



Looks good; let's also try when truncated to 3 categories

In [425]:
```
nclust = 3
df3['|sim3_img0-2-3|'] = df3['sim3_img0-2-3'].map(lambda x: int( ((x+1)/2) * nclust  - 0.0001))
df3['|sim3_img0-2-3|'].value_counts()

imgutils.show_large_heatmap(df3, '|sim3_img0-2-3|', imgnames[0:6], n_rows=2, n_cols=3, fig_size=figsize)
```

Score (silhouette)

In [423]:
```
# assess the clustering score in 'original space' and 'pca space':
print_score('PCA Similarity (original)', X, df3['|sim3_img0-2-3|'] )
print_score('PCA Similarity (pca)', X_pca, df3['|sim3_img0-2-3|'] )
print_score('PCA Similarity (normalized pca)', X_pca_norm, df3['|sim3_img0-2-3|'] )
```

```
PCA Similarity (original): 0.425053
PCA Similarity (pca): 0.425067
PCA Similarity (normalized pca): 0.380479
```

So, with similarity approach to cluster into 3 groups, gives a clustering score of ~ 0.42 in original or PCA space

---

10. Conclusions

- Via extra study and the DataCamp courses, I learned a few new techniques which I utilized here.
- One of them is a scoring based on the clusters via **silhouette scoring** , which does not require labelling
- An alternate unsupervised learning technique is **NMF (Non-negative Matrix Factorization)** plus feature vector similarity; (often this is used for texts or for images (but than based on the pixel values)
- The **NMF approach did not work** well for this data set
- However, using the same **similarity vector** approach **on original statistics works well** (on this dataset)
- Applying the similarity approach on the PCA transformed statistics gave similar results

In other words: **(simple) vector similarity can be a good alternative for the heat maps**

11. Next Steps:

- try whole pipeline on the harder set
- consider chaining steps together via sklearn.pipeline

Michael Janus, 16 August 2018