Full Pipeline (on Tileset6) - Oct 2018

Created: 21 Aug 2018
Last update: 5 Nov 2018

Goal: Run the full pipeline on a harder set

---

1. Imports

In [2]:
```
# 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.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.metrics import silhouette_score

import imgutils
```

In [3]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)
```

Out[3]:
```
<module 'imgutils' from 'C:\\JADS\\SW\\Grad Proj\\sources\\imgutils.py'>
```

---

2. Data Definitions & Feature Specification

In [4]:
```
# Data:
datafolder = '../data/Crystals_Apr_12/Tileset6_Subset_Blacks_2k'
n_tiles_x = 4  # mostly for visualization
n_tiles_y = 2


# Features to use:
#feature_funcs = [imgutils.img_mean, imgutils.img_std, imgutils.img_median,
#                 imgutils.img_mode,
#                 imgutils.img_kurtosis, imgutils.img_skewness]
feature_funcs = [imgutils.img_std, imgutils.img_relstd, imgutils.img_mean,
                 imgutils.img_skewness,  imgutils.img_kurtosis, imgutils.img_mode, imgutils.img_range]
feature_names = imgutils.stat_names(feature_funcs)

# Size of the grid, specified as number of slices per image in x and y direction:
default_grid_x = 4
default_grid_y = default_grid_x
```

---

3. Import Data & Extract Features

In [5]:
```
# image import:
print("Scanning for images in '{}'...".format(datafolder))
df_imgfiles = imgutils.scanimgdir(datafolder, '.tif')
imgfiles = list(df_imgfiles['filename'])
print("# of images: {} \n".format(len(imgfiles)))

# feature extraction:
print("Feature extraction...")
print("- Slicing up images in {} x {} patches. ".format(default_grid_y, default_grid_x))
print("- Extract statistics from each slice: {} ".format(', '.join(feature_names)))
print("...working...", end='\r')
df = imgutils.slicestats(imgfiles, default_grid_y, default_grid_x, feature_funcs)
print("# slices extracted: ", len(df))
```

```
Scanning for images in '../data/Crystals_Apr_12/Tileset6_Subset_Blacks_2k'...
# of images: 8

Feature extraction...
- Slicing up images in 4 x 4 patches.
- Extract statistics from each slice: img_std, img_relstd, img_mean, img_skewness, img_kurtosis, img_mode, img_range
# slices extracted:  128
```

---

4. Machine Learning Pipeline

Hyper parameters

In [6]:
```
# data hyper-parameters
default_n_clusters = 3

# algorithm hyper-parameters:
kmeans_n_init = 10
```

In [7]:
```
def run_ml_pipeline2(X, ml_name, ml_algorithm, standardize=True, use_pca=True, n_pca=None):

    # Setup algorithmic pipeline, including standardization
    pipeline = Pipeline([(ml_name, ml_algorithm)])

    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=n_pca)))
    if (standardize):
        pipeline.steps.insert(0, ('scaling_{0}'.format(ml_name), StandardScaler()))

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [8]:
```
def run_ml_pipelines2(df_data, feature_cols, n_clusters, standardize=True, use_pca=True, n_pca=None):
    global kmeans_n_init

    X = df_data.loc[:,feature_cols]

    # Setup ML clustering algorithms:
    kmeans = KMeans(algorithm='auto', n_clusters=n_clusters, n_init=kmeans_n_init, init='k-means++')
    agglomerative =  AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete')

    # run the pipelines
    print("Executing clustering pipelines...")
    score_kmeans, y_kmeans = run_ml_pipeline2(X, 'kmeans', kmeans, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    score_hier, y_hier = run_ml_pipeline2(X, 'hierarchical', agglomerative, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    print("Done\n")

    # collect data
    df_data['kmeans']=y_kmeans
    df_data['hierarchical']=y_hier

    # report results:
    print("\nClustering Scores:")
    print("K-means: ", score_kmeans)
    print("Hierarchical: ", score_hier)
```

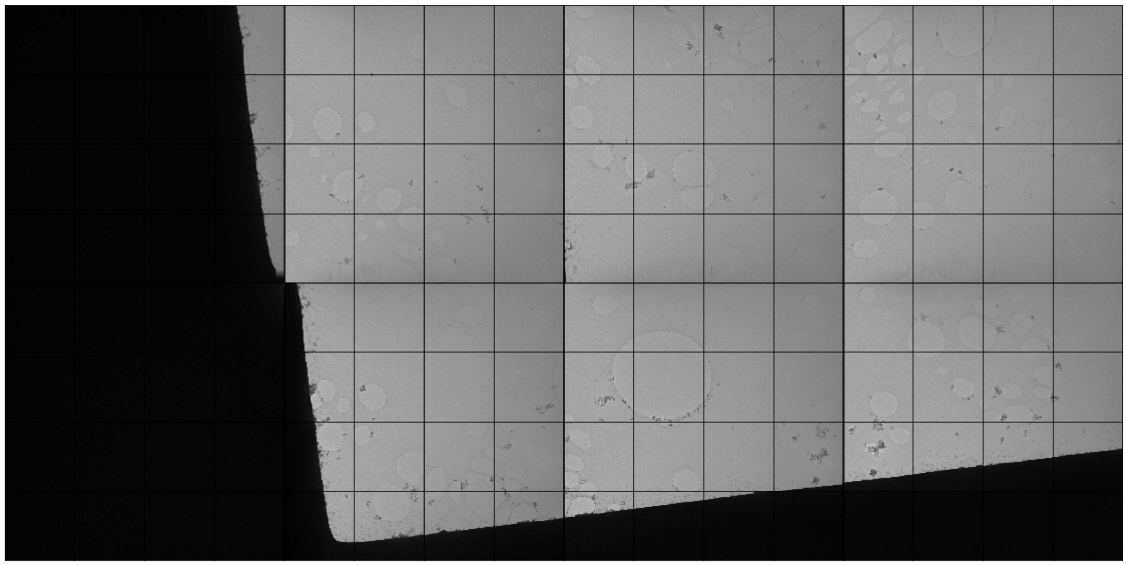
In [9]:
```
run_ml_pipelines2(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.7235104512504366
Hierarchical:  0.23921655669437558
```

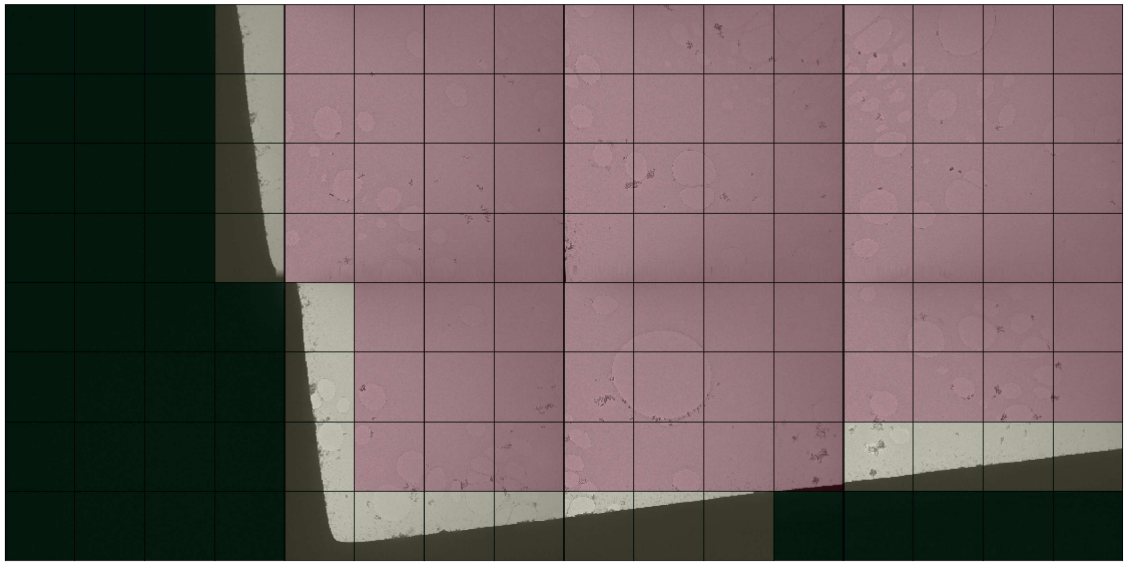
In [10]:
```
df['dummy'] = 0
imgutils.show_large_heatmap(df, 'dummy', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,10))
```

Heats from: dummy



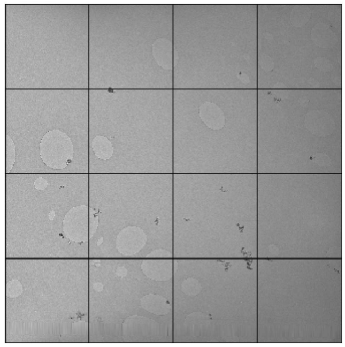
In [17]:
```
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,10))
```

Heats from: kmeans



In [18]:
```
imgfile1 = imgfiles[1]
#df_heats1 = df[df['filename']==imgfile1]['kmeans']
#heats = np.reshape(df_heats1.values, (default_grid_y, default_grid_x))

subimgs, heats = imgutils.getimgslices_fromdf(df, imgfile1, 'kmeans')
heats = heats / np.max(heats)
imgutils.showheatmap(subimgs, heats, cmapname='Set1', heatdepend_opacity = False)
```



In [19]:
```
n_patches = 40
df2 = imgutils.slicestats([imgfile1], n_patches, n_patches, feature_funcs)
```

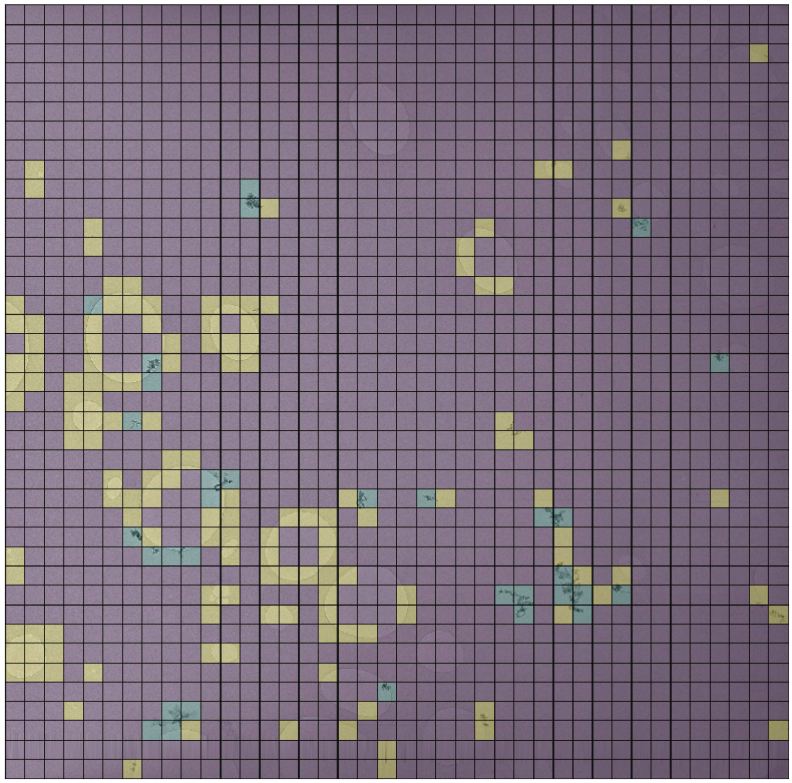
In [23]:
```
run_ml_pipelines2(df2, ['img_std', 'img_range'], 3, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.7420027111971942
Hierarchical:  0.8170861503218958
```

In [34]:
```
imgutils.show_large_heatmap(df2, 'kmeans', [imgfile1], n_rows=1, n_cols=1, fig_size=(14,14), cmapname='viridis', opacity=0.25)
```

Heats from: kmeans



In [ ]:
```
print(default_n_clusters)
```

Use other scoring

Adjusting the ml_pipeline to use silhouette scoring based on it's last transformation: (later renamed the other ones to run_xxx2 to preserve them)

In [20]:
```
from sklearn.metrics import calinski_harabaz_score

def run_ml_pipeline(X, ml_name, ml_algorithm, standardize=True, use_pca=True, n_pca=None):

    # Setup 'manual' pipeline (not using sklearn pipeline as intermediates are needed)
    feat_data = X
    if (standardize):
        standardizer = StandardScaler()
        X_norm = standardizer.fit_transform(X)
        feat_data = X_norm
    if (use_pca):
        pca = PCA(n_components=n_pca)
        X_pca = pca.fit_transform(feat_data)
        feat_data = X_pca

    # run the pipelines
    y = ml_algorithm.fit_predict(feat_data) # calls predict oto get the labels

    # report score:
    #score = silhouette_score(feat_data, y)
    score = calinski_harabaz_score(feat_data,y)

    return score, y
```

In [21]:
```
def run_ml_pipelines(df_data, feature_cols, n_clusters, standardize=True, use_pca=True, n_pca=None):
    global kmeans_n_init

    X = df_data.loc[:,feature_cols]

    # Setup ML clustering algorithms:
    kmeans = KMeans(algorithm='auto', n_clusters=n_clusters, n_init=kmeans_n_init, init='k-means++')
    agglomerative =  AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete')

    # run the pipelines
    print("Executing clustering pipelines...")
    score_kmeans, y_kmeans = run_ml_pipeline(X, 'kmeans', kmeans, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    score_hier, y_hier = run_ml_pipeline(X, 'hierarchical', agglomerative, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    print("Done\n")

    # collect data
    df_data['kmeans']=y_kmeans
    df_data['hierarchical']=y_hier

    # report results:
    print("\nClustering Scores:")
    print("K-means: ", score_kmeans)
    print("Hierarchical: ", score_hier)
```

In [22]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  116.78546926005859
Hierarchical:  77.16127883477874
```

More consistent with previous results and imo it is indeed better to assess the algorithm on how good it could cluster after all pre-processing

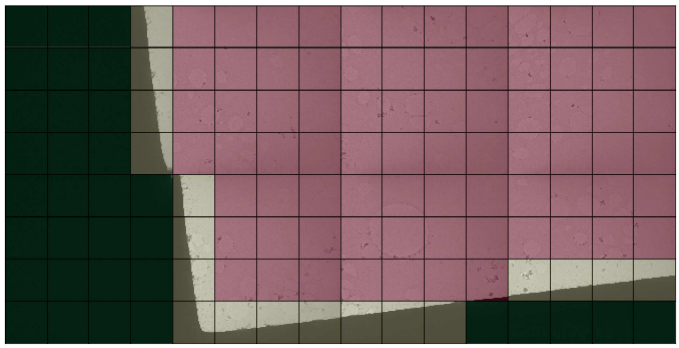
5. Visualize results

In [23]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
s = (8,6)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

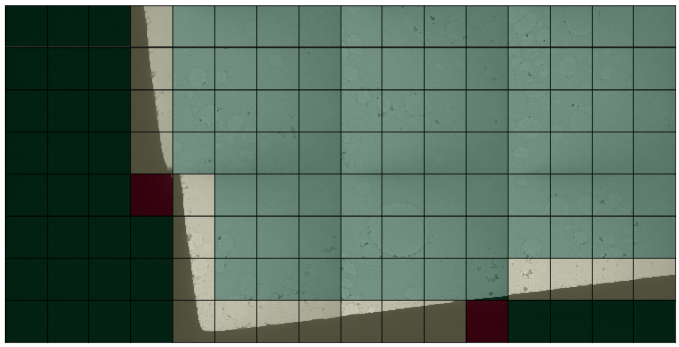
```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  116.78546926005859
Hierarchical:  77.16127883477874
```

Heats from: kmeans



Heats from: hierarchical



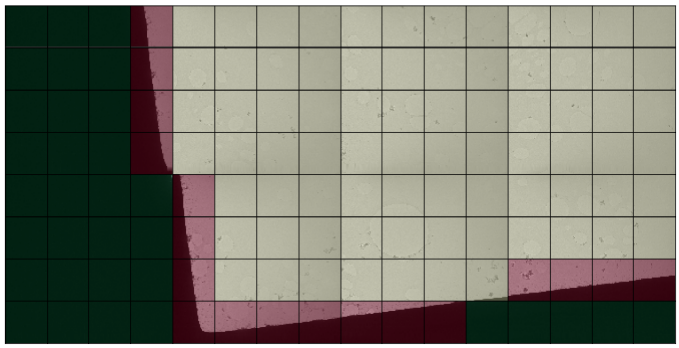
Run it again without PCA and/pr normalization compare results

In [24]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=False)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

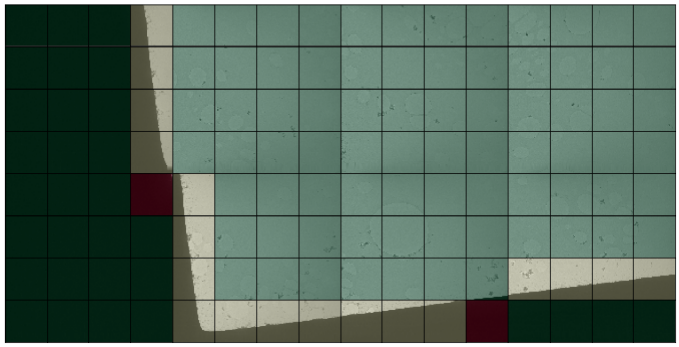
```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  116.78546926005839
Hierarchical:  77.16127883477883
```

Heats from: kmeans



Heats from: hierarchical

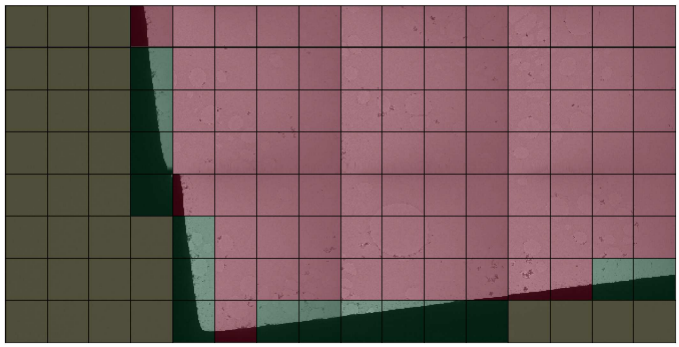


In [25]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=False, use_pca=False)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

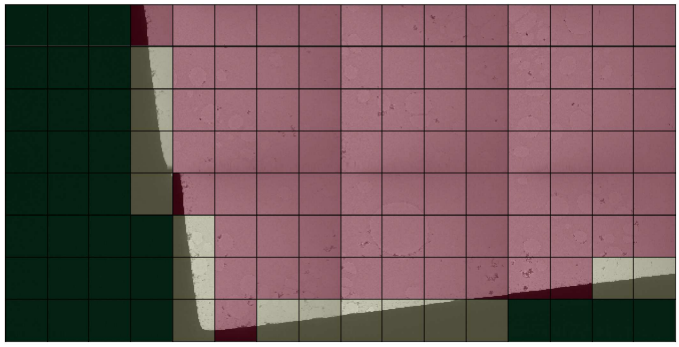
```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  430.54014744225793
Hierarchical:  430.54014744225793
```

Heats from: kmeans



Heats from: hierarchical



On this dataset, not much difference between hierarchical and pca, with or without normalization

6. Combine import and pipeline:

In [26]:
```
def import_data(imagefolder):
    df_imgfiles = imgutils.scanimgdir(imagefolder, '.tif')
    return list(df_imgfiles['filename'])

def extract_features(imgfiles, feature_funcs, n_grid_rows, n_grid_cols):
    df = imgutils.slicestats(imgfiles, n_grid_rows, n_grid_cols, feature_funcs)
    feature_names = imgutils.stat_names(feature_funcs)
    return df, feature_names
```

In [27]:
```
def run_kmeans_pipeline(df_data, feature_cols, n_clusters, standardize=True, use_pca=True, n_pca= None):
    global kmeans_n_init

    ml_name="kmeans"
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clusters, n_init=kmeans_n_init, init='k-means++')

    X = df_data.loc[:,feature_cols]
    score, y = run_ml_pipeline(X, ml_name, ml_algorithm, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    df_data[ml_name]= y

    return score

def run_hierarchical_pipeline(df_data, feature_cols, n_clusters, standardize=True, use_pca=True, n_pca=None):

    ml_name="hierarchical"
    ml_algorithm =  AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete')

    X = df_data.loc[:,feature_cols]
    score, y = run_ml_pipeline(X, ml_name, ml_algorithm, standardize = standardize, use_pca = use_pca, n_pca=n_pca)
    df_data[ml_name]= y

    return score
```

In [28]:
```
def run_fullpipeline(imagefolder, n_image_rows, n_image_cols,
                     n_grid_rows, n_grid_cols, feature_funcs, n_clusters, fig_size=(8,6), return_df = False):
    """
    Run the full pipeline from import to visualization.
    """
    print("Working...\r")
    imgfiles = import_data(imagefolder)
    df, feature_names = extract_features(imgfiles, feature_funcs, n_grid_rows, n_grid_cols)
    print(feature_names)
    score_kmeans = run_kmeans_pipeline(df, feature_names, n_clusters, standardize=True, use_pca=True )
    score_hier = run_hierarchical_pipeline(df, feature_names, n_clusters, standardize=False, use_pca=False)

    print('Results:')
    print('Score k-means:', score_kmeans)
    print('Score hierarchical:', score_hier)

    print('Visualizing...')
    imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)
    imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)

    if return_df: return df
```

In [29]:
```
def run_fullpipeline_kmeans(imagefolder, n_image_rows, n_image_cols,
                     n_grid_rows, n_grid_cols, feature_funcs, n_clusters, fig_size=(8,6), return_df = False):
    """
    Run the full pipeline from import to visualization.
    """
    print("Working...\r")
    imgfiles = import_data(imagefolder)
    df, feature_names = extract_features(imgfiles, feature_funcs, n_grid_rows, n_grid_cols)
    print(feature_names)
    score_kmeans = run_kmeans_pipeline(df, feature_names, n_clusters, standardize=True, use_pca=True )

    print('Results:')
    print('Score k-means:', score_kmeans)

    print('Visualizing...')
    imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)

    if return_df: return df
```

In [30]:
```
def run_fullpipeline_hierarchical(imagefolder, n_image_rows, n_image_cols,
                     n_grid_rows, n_grid_cols, feature_funcs, n_clusters, fig_size=(8,6), return_df = False):
    """
    Run the full pipeline from import to visualization.
    """
    print("Working...\r")
    imgfiles = import_data(imagefolder)
    df, feature_names = extract_features(imgfiles, feature_funcs, n_grid_rows, n_grid_cols)
    print(feature_names)
    score_hier = run_hierarchical_pipeline(df, feature_names, n_clusters, standardize=False, use_pca=False)

    print('Results:')
    print('Score hierarchical:', score_hier)

    print('Visualizing...')
    imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)

    if return_df: return df
```

7. Try it out with different combinations of slices

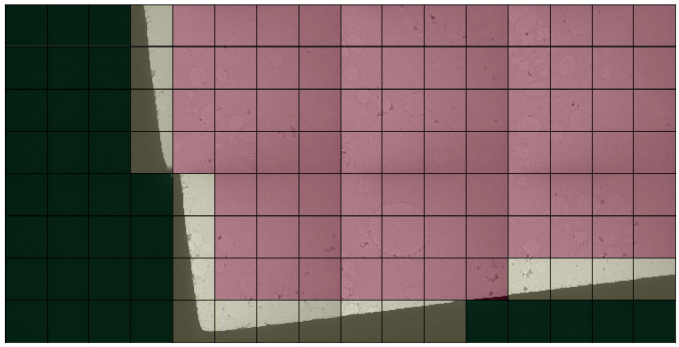
In [31]:
```
datafolder = '../data/Crystals_Apr_12/Tileset6_Subset_Blacks_1k'
```

4x4 - 3 clusters

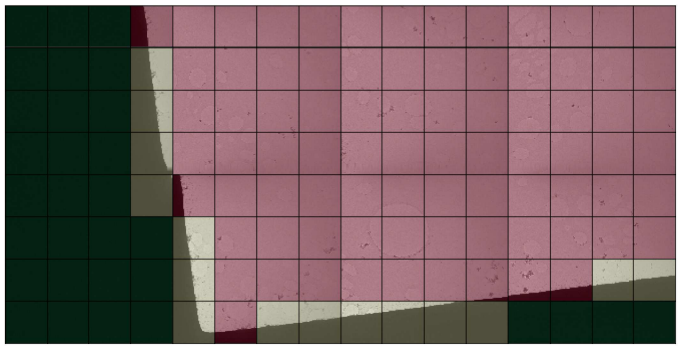
In [32]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 3)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 116.05834163794131
Score hierarchical: 416.2049667862349
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical

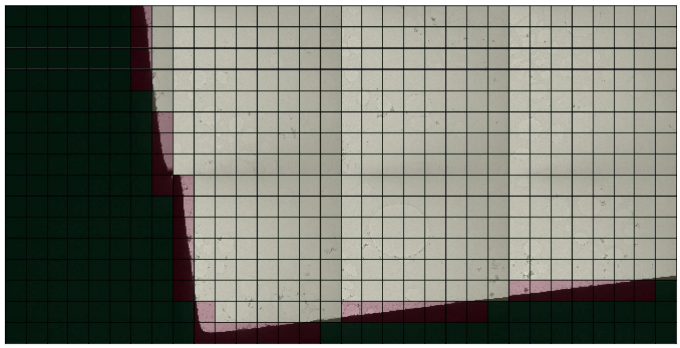


8x8, 3 clusters

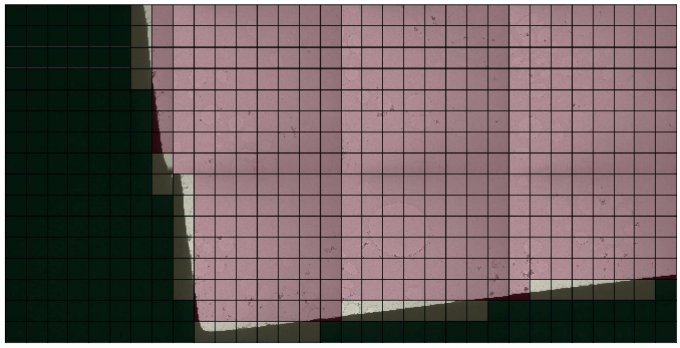
In [49]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 3)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 461.76510133329776
Score hierarchical: 1927.0963309363449
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical

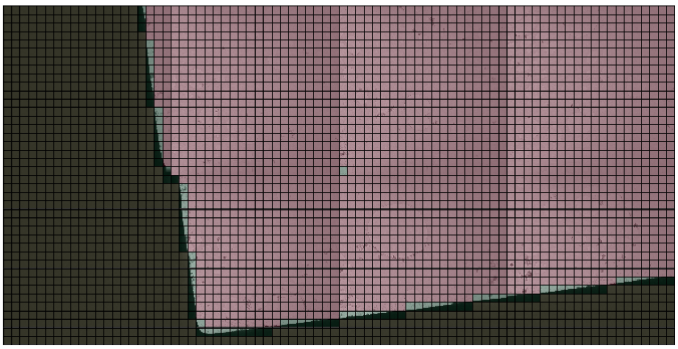


20x20, 3 clusters

In [50]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 20, 20, feature_funcs, 3)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 2546.8860096894487
Score hierarchical: 17063.445015093297
Visualizing...
```

Heats from: kmeans



```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-50-66c83d58066b> in <module>()
----> 1 run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 20, 20, feature_funcs, 3)

<ipython-input-44-d37503be57f0> in run_fullpipeline(imagefolder, n_image_rows, n_image_cols, n_grid_rows, n_grid_cols, feature_funcs, n_clusters, fig_size, return_df)
     17     print('Visualizing...')
     18     imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)
---> 19     imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=fig_size)
     20
     21     if return_df: return df

C:\JADS\SW\Grad Proj\sources\imgutils.py in show_large_heatmap(df_imgstats, heatcolname, imgnames, n_rows, n_cols, opacity, cmapname, heatdependent_opacity, fig_size, a
nnotate_tiles, show_extra_info, return_heatmap, subtitle, no_borders, relspacing)
    284     if (subtitle != None): tittxt += " - " + subtitle
    285     showheatmap(allsubimgs, allheats, heatdepend_opacity=heatdependent_opacity, opacity=opacity, cmapname=cmapname,
--> 286                 title=tittxt, figsize=fig_size, tile_annotations=allannos, no_borders=no_borders, relspacing=relspacing )
    287
    288     # show info if requested

C:\JADS\SW\Grad Proj\sources\imgutils.py in showheatmap(imgs, heats, cmapname, opacity, heatdepend_opacity, title, figsize, tile_labels, tile_annotations, no_borders, r
elspacing)
    183     # adjust fig size (height is leading) to avoid white space between images
    184     figsize2 = (nx * figsize[1]/ny, figsize[1]+0.2)  # height is leading
--> 185     fig, subfigs = plt.subplots(ny,nx, sharex=False, sharey=False, figsize = figsize2)
    186     graymin, graymax = getimgs_minmax(imgs)
    187

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\pyplot.py in subplots(nrows, ncols, sharex, sharey, squeeze, subplot_kw, gridspec_kw, **fig_kw)
   1199     axs = fig.subplots(nrows=nrows, ncols=ncols, sharex=sharex, sharey=sharey,
   1200                        squeeze=squeeze, subplot_kw=subplot_kw,
-> 1201                        gridspec_kw=gridspec_kw)
   1202     return fig, axs
   1203

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in subplots(self, nrows, ncols, sharex, sharey, squeeze, subplot_kw, gridspec_kw)
   1349                 subplot_kw["sharex"] = shared_with[sharex]
   1350                 subplot_kw["sharey"] = shared_with[sharey]
-> 1351                 axarr[row, col] = self.add_subplot(gs[row, col], **subplot_kw)
   1352
   1353         # turn off redundant tick labeling

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in add_subplot(self, *args, **kwargs)
   1239             a = subplot_class_factory(projection_class)(self, *args, **kwargs)
   1240         self._axstack.add(key, a)
-> 1241         self.sca(a)
   1242         a._remove_method = self.__remove_ax
   1243         self.stale = True

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in sca(self, a)
   1819     def sca(self, a):
   1820         """Set the current axes to be a and return a."""
-> 1821         self._axstack.bubble(a)
   1822         for func in self._axobservers:
   1823             func(self)

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in bubble(self, a)
    118         stack, to the top.
    119         """
--> 120         return Stack.bubble(self, self._entry_from_axes(a))
    121
    122     def add(self, key, a):

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\cbook\__init__.py in bubble(self, o)
   1244                 bubbles.append(thiso)
   1245             else:
-> 1246                 self.push(thiso)
   1247         for thiso in bubbles:
   1248             self.push(o)

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\cbook\__init__.py in push(self, o)
   1212         self._elements.append(o)
   1213         self._pos = len(self._elements) - 1
-> 1214         return self()
   1215
   1216     def home(self):

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in __call__(self)
    164
    165     def __call__(self):
--> 166         return self.current_key_axes()[1]
    167
    168     def __contains__(self, a):

C:\ProgramData\Anaconda3\envs\JADS\lib\site-packages\matplotlib\figure.py in current_key_axes(self)
    157
    158         """
--> 159         if not len(self._elements):
    160             return self._default, self._default
    161         else:

KeyboardInterrupt:
```

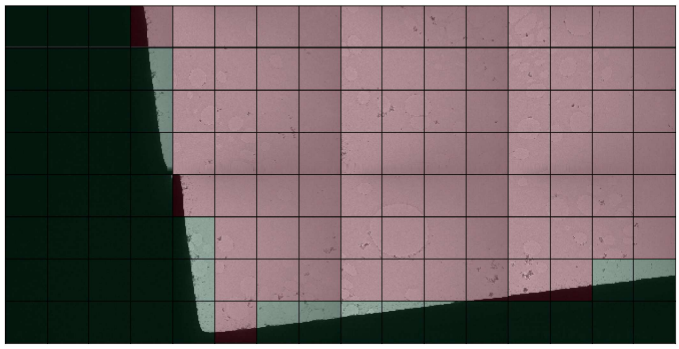
8. Try it out with different number of clusters

2 clusters (4x4 , 10x10, 20x20)

In [56]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 2)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 67.41728307925776
Visualizing...
```

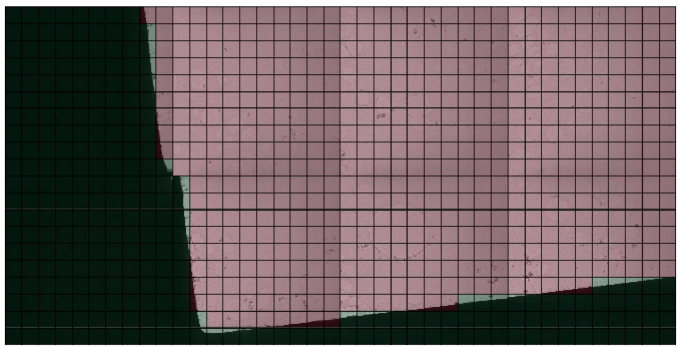
Heats from: kmeans



In [57]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 10, 10, feature_funcs, 2)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 437.7566150915102
Visualizing...
```

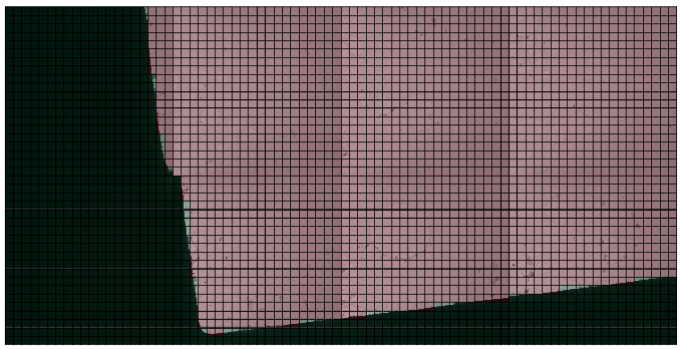
Heats from: kmeans



In [59]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 20, 20, feature_funcs, 2)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 1596.1667325824951
Visualizing...
```

Heats from: kmeans

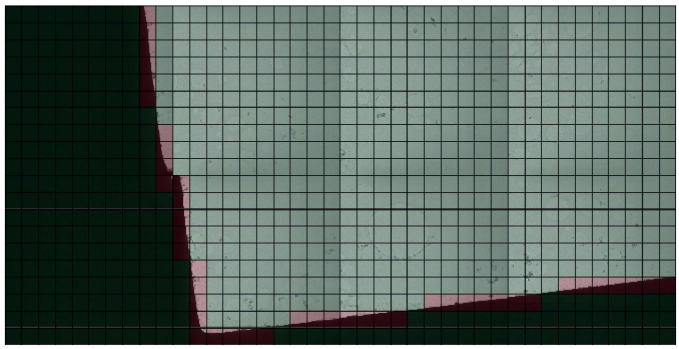


Look again at hierarchical with scaling and pca on:

In [60]:
```
imgfiles = import_data(datafolder)
df_temp, feature_names = extract_features(imgfiles, feature_funcs, 10, 10)
score = run_hierarchical_pipeline(df_temp, feature_names, 2, standardize=True, use_pca=True)
print("With scaling & pca:", score)
imgutils.show_large_heatmap(df_temp, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
score = run_hierarchical_pipeline(df_temp, feature_names, 2, standardize=False, use_pca=False)
print("Without scaling & pca:", score)
imgutils.show_large_heatmap(df_temp, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

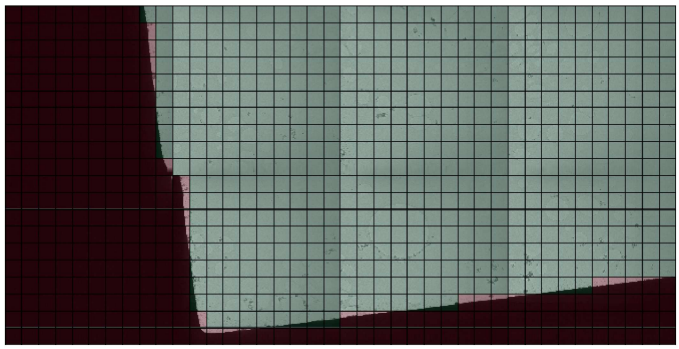
```
With scaling & pca: 360.8157196857184
```

Heats from: hierarchical



```
Without scaling & pca: 3034.45384899735
```

Heats from: hierarchical



In [ ]:
```
 imgfiles = import_data(datafolder)
df_temp, feature_names = extract_features(imgfiles, feature_funcs, 6, 6)
score = run_kmeans_pipeline(df_temp, feature_names, 2, standardize=True, use_pca=True)
print("With scaling & pca:", score)
imgutils.show_large_heatmap(df_temp, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
score = run_kmeans_pipeline(df_temp, feature_names, 2, standardize=False, use_pca=False)
print("Without scaling & pca:", score)
imgutils.show_large_heatmap(df_temp, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

In [69]:
```
print(feature_funcs)
```

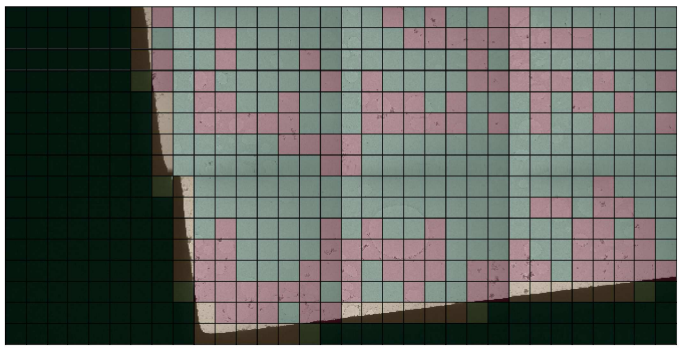
```
[<function img_std at 0x000000000BC017B8>, <function img_relstd at 0x000000000BC01730>, <function img_mean at 0x000000000BC01158>, <function img_skewness at 0x000000000
BC020D0>, <function img_kurtosis at 0x000000000BC02048>, <function img_mode at 0x000000000BC01E18>, <function img_range at 0x000000000BC012F0>]
```

no difference, all not good. Let's try if we drop the mean and mode (see if ignores the black)

In [72]:
```
try_funcs =  [imgutils.img_std, imgutils.img_relstd, imgutils.img_skewness,imgutils.img_kurtosis]
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 8, 8, try_funcs , 4)
```

```
Working...
['img_std', 'img_relstd', 'img_skewness', 'img_kurtosis']
Results:
Score k-means: 942.6569420097127
Visualizing...
```

Heats from: kmeans

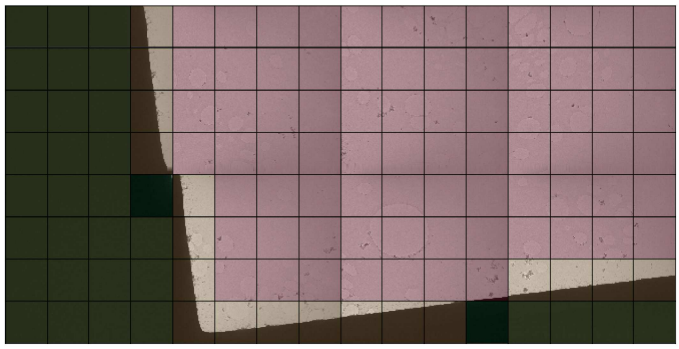


4 clusters (4x4, 10x10, 20x20)

In [61]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 4)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 246.29078753065716
Visualizing...
```

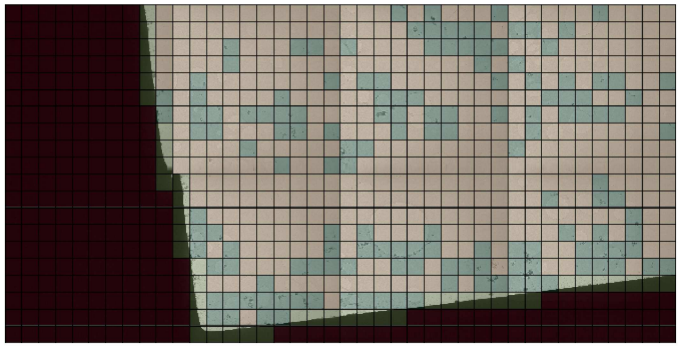
Heats from: kmeans



In [62]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 10, 10, feature_funcs, 4)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 1425.3953592594607
Visualizing...
```

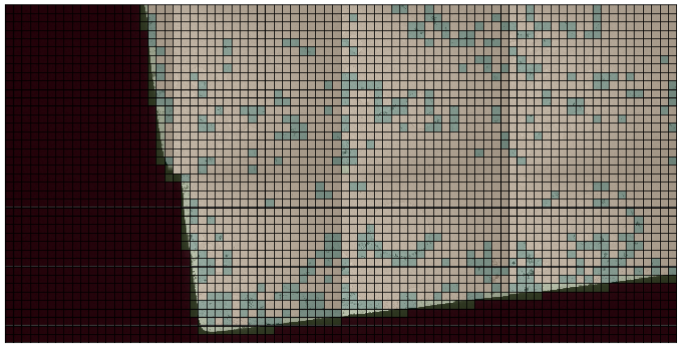
Heats from: kmeans



In [63]:
```
run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 20, 20, feature_funcs, 4)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 3665.6469480762325
Visualizing...
```

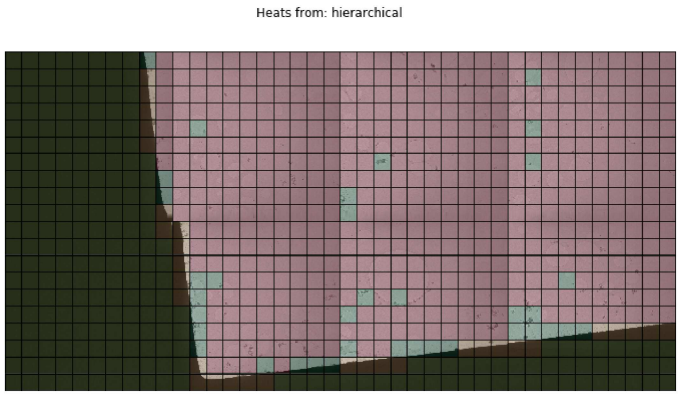
Heats from: kmeans



with smaller grid and 4 clusters, it starts to make sense

In [75]:
```
run_fullpipeline_hierarchical(datafolder, n_tiles_y, n_tiles_x, 10, 10, feature_funcs, 4)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score hierarchical: 3377.35144128575
Visualizing...
```



Need 'grid search' versions to scan parameter space!!!

9. (Grid) Search for hyper-parameter optimizations

In [ ]:
```
imgfiles = import_data(datafolder)
df, feature_names = extract_features(imgfiles, feature_funcs, default_grid_y, default_grid_x)
```

9A. Number of clusters

In [ ]:
```
def optimize_kmeans_nclusters(df_imgstats, feature_names, n_clusters_start, n_clusters_end):
    """
    Run the full pipeline (except for visualization) for different number of clusters
    and show graph with score vs number of clusters.
    """
    n_count = n_clusters_end + 1 - n_clusters_start
    result = np.empty(shape=(0,2), dtype=float)

    for n in range(n_clusters_start, n_clusters_end+1):
        score = run_kmeans_pipeline(df, feature_names, n, standardize=True, use_pca=True )
        result = np.append(result, np.array([[n, score]]), axis=0)

    plt.plot(result[:,0], result[:,1])
    plt.title('K-means Clustering - Score vs. Number of Clusters ')
    plt.xlabel('# Clusters')
    plt.ylabel('Score')
    plt.show()
```

In [ ]:
```
def optimize_hierarchical_nclusters(df_imgstats, feature_names, n_clusters_start, n_clusters_end):
    """
    Run the full pipeline (except for visualization) for different number of clusters
    and show graph with score vs number of clusters.
    """
    n_count = n_clusters_end + 1 - n_clusters_start
    result = np.empty(shape=(0,2), dtype=float)

    for n in range(n_clusters_start, n_clusters_end+1):
        score = run_hierarchical_pipeline(df, feature_names, n, standardize=False, use_pca=False )
        result = np.append(result, np.array([[n, score]]), axis=0)

    plt.plot(result[:,0], result[:,1])
    plt.title('Hierarchical Clustering - Score vs. Number of Clusters ')
    plt.xlabel('# Clusters')
    plt.ylabel('Score')
    plt.show()
```

In [ ]:
```
optimize_kmeans_nclusters(df, feature_names, 2, 6)
```

In [ ]:
```
optimize_hierarchical_nclusters(df, feature_names, 2, 6)
```

9B. Number of features / PCA Components

In [ ]:
```
def optimize_kmeans_pcacomponents(df_imgstats, feature_names, n_clusters):
    """
    Run the full pipeline (except for visualization) for different number of PCA componenents
    and show graph with the score vs number of components.
    """
    n_count = len(feature_names) - 1
    result = np.empty(shape=(0,2), dtype=float)

    for n in range(1, len(feature_names)):
        score = run_kmeans_pipeline(df, feature_names, n_clusters, standardize=True, use_pca=True, n_pca = n )
        result = np.append(result, np.array([[n, score]]), axis=0)

    plt.plot(result[:,0], result[:,1])
    plt.title('K-means Clustering - Score vs. Number of PCA components ')
    plt.xlabel('N PCA Components')
    plt.ylabel('Score')
    plt.show()
```

In [ ]:
```
def optimize_hierarchical_pcacomponents(df_imgstats, feature_names, n_clusters):
    """
    Run the full pipeline (except for visualization) for different number of PCA componenents
    and show graph with the score vs number of components.
    """
    n_count = len(feature_names) - 1
    result = np.empty(shape=(0,2), dtype=float)

    for n in range(1, len(feature_names)):
        score = run_hierarchical_pipeline(df, feature_names, n_clusters, standardize=True, use_pca=True, n_pca = n )
        result = np.append(result, np.array([[n, score]]), axis=0)

    plt.plot(result[:,0], result[:,1])
    plt.title('Hierarchical Clustering - Score vs. Number of PCA components ')
    plt.xlabel('N PCA Components')
    plt.ylabel('Score')
    plt.show()
```

In [ ]:
```
optimize_kmeans_pcacomponents(df, feature_names, n_clusters=3)
```

Interesting. This graph says that a single component (after PCA transform) would already do a good job. Let's visualize

In [ ]:
```
score = run_kmeans_pipeline(df, feature_names, 3, standardize=True, use_pca=True, n_pca=1)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(10,8))
```

Indeed not bad. Maybe not so surprise as in early EDA we saw that on this particular set the standard deviation is very informative (we used it for manual labelling)

A 'manual grid search' showed for 4 clusters the same

In [ ]:
```
optimize_kmeans_pcacomponents(df, feature_names, 4)
score = run_kmeans_pipeline(df, feature_names, 4, standardize=True, use_pca=True, n_pca=1)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(10,8))
```

Actually this looks very good! It properly grouped 'fully occupied grid cells' from 'partially occupied'!

Let's also look at hierarchical clustering (although scores indicated better results without PCA)

In [ ]:
```
optimize_hierarchical_pcacomponents(df, feature_names, 2)
optimize_hierarchical_pcacomponents(df, feature_names, 3)
optimize_hierarchical_pcacomponents(df, feature_names, 4)
```

Don't know what these graphs really say; as the score is assessing cluster cohesion based on the last transformation in the pre-processing, it can be expected that less components are more separated. Maybe need to look at the 'elbows' here?

In [ ]:
```
# visualize for 4 clusters with 1 pca component:
```

In [ ]:
```
score = run_hierarchical_pipeline(df, feature_names, 4, standardize=True, use_pca=True, n_pca=1)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(10,8))
```

Still looks like pretty good grouping into 4 clusters!

10. (Grid) Search for number of patches (expensive)

As the image slicing and feature extraction are the computationally expensive parts, both algorithms will be run in the search

10A. Optimize and plot graphs

In [ ]:
```
import sys

def optimize_slices(imagefolder, feature_funcs, n_clusters, list_n_slices):
    """
    Run the full pipeline (without visualization) for different number of image slices
    and show graph of score vs number of slices.
    Note that same number of slices is used in x and y direction.
    Returns a list of tuples of form (dataframe, n_slices, score_kmeans, score_hierarchical)
    """
    imgfiles = import_data(imagefolder)
    results = []

    print("Running slice optimization for {} clusters:".format(n_clusters))
    counter=0
    for n_slices in list_n_slices:
        progress = 100 * counter / len(list_n_slices)
        sys.stdout.write("{:.1f} %".format(progress) + "\r")

        df, feature_names = extract_features(imgfiles, feature_funcs, n_grid_rows=n_slices, n_grid_cols=n_slices)
        score_kmeans = run_kmeans_pipeline(df, feature_names, n_clusters, standardize=True, use_pca=True )
        score_hier = run_hierarchical_pipeline(df, feature_names, n_clusters, standardize=False, use_pca=False)
        results.append((df, n_slices, score_kmeans, score_hier))

        counter += 1

    sys.stdout.write("Done.")
    #sys.stdout.flush()

    plotdata = list(zip(*results))

    plt.plot(plotdata[1], plotdata[2], label="k-means (with PCA)")
    plt.plot(plotdata[1], plotdata[3], label="hierarchical")
    plt.title('Clustering - Score vs. Number of Image Slices')
    plt.xlabel('Number of slices')
    plt.ylabel('Score')
    plt.legend(loc='upper right')
    plt.show()

    return results
```

In [ ]:
```
slice_opt_results_3clusters = optimize_slices(datafolder, feature_funcs, n_clusters=3, list_n_slices=[2,4,8,10,12,16])
```

In [ ]:
```
slice_opt_results_4clusters = optimize_slices(datafolder, feature_funcs, n_clusters=4, list_n_slices=[2,4,8,10,12,16,20,32])
```

Interesting, optimum number of slices is around 10 for 3 clusters, and even higher number of patches for 4 clusters.

Also it is more efficient to do a **grid search** for clusters vs patches, as cluster evaluation is cheap while patches is expensive.

In [ ]:
```
import sys

def gridsearch_slices(imagefolder, feature_funcs, list_n_clusters, list_n_slices):
    """
    Run the full pipeline (without visualization) for different number of image slices
    and show graph of score vs number of slices.
    Note that same number of slices is used in x and y direction.
    Returns a list of tuples of form (dataframe, n_slices, n_clusters, score_kmeans, score_hierarchical)
    """
    imgfiles = import_data(imagefolder)
    results = []

    print("Running slice-#clusters grid-search...")
    counter=0
    for n_slices in list_n_slices:
        progress = 100 * counter / len(list_n_slices)
        sys.stdout.write("{:.1f} %".format(progress) + "\r")

        df, feature_names = extract_features(imgfiles, feature_funcs, n_grid_rows=n_slices, n_grid_cols=n_slices)

        for n_clust in list_n_clusters:
            df2 = df.copy()   # each result should have its own dataframe otherwise results are overwritten
            score_kmeans = run_kmeans_pipeline(df2, feature_names, n_clust, standardize=True, use_pca=True )
            score_hier = run_hierarchical_pipeline(df2, feature_names, n_clust, standardize=False, use_pca=False)
            results.append((df2, n_slices, n_clust, score_kmeans, score_hier))

        counter += 1

    sys.stdout.write("Done.")

    for n_clust in list_n_clusters:
        results_for_cluster = [t for t in results if t[2]==n_clust]
        plotdata = list(zip(*results_for_cluster))

        plt.plot(plotdata[1], plotdata[3], label="k-means (with PCA)")
        plt.plot(plotdata[1], plotdata[4], label="hierarchical")
        plt.title('{} Clusters - Score vs. Number of Image Slices'.format(n_clust))
        plt.xlabel('Number of slices')
        plt.ylabel('Score')
        plt.legend(loc='upper right')
        plt.show()


    return results
```

In [ ]:
```
slice_grid_search = gridsearch_slices(datafolder, feature_funcs, list_n_clusters=[2,3,4,5,6], list_n_slices=[2,4,6,8,10,12,14,16,20,32])
```

(I one time ran it up to 64 slices, but that take forever. k-means flattens and hierarchical goes drops drastically)

10B. Look at some heatmaps for 2 clusters

In [ ]:
```
def get_df_from_slicegridsearch(results, num_clust, num_slices):
    results_for_cluster = [t for t in results if t[2]==num_clust]
    item_in_list = [t for t in results_for_cluster if t[1]==num_slices][0]
    return item_in_list[0]

def show_from_slicegridsearch(results, num_clust, num_slices, n_img_rows, n_img_cols, heat_col='kmeans', fig_size=(12,10)):
    df_toshow = get_df_from_slicegridsearch(results, num_clust, num_slices)
    imgfiles = df_toshow['filename'].unique().tolist()
    extratitle = "{}x{} slices, {} clusters".format(num_slices,num_slices, num_clust)
    imgutils.show_large_heatmap(df_toshow, heat_col, imgfiles, n_rows=n_img_rows, n_cols=n_img_cols, fig_size=fig_size, subtitle=extratitle)
```

In [ ]:
```
def show_multiple_from_slicegridsearch(results, list_num_clust, list_num_slices, n_img_rows, n_img_cols,
                                       heat_col='kmeans', fig_size=(12,10)):
    for n_slices in list_num_slices:
        for n_clust in list_num_clust:
            show_from_slicegridsearch(results, n_clust, n_slices, n_img_rows, n_img_cols, heat_col,  fig_size=fig_size)
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [2], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(10,8))
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [2], [2,4,8, 10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical',  fig_size=(10,8))
```

10c. Look at some heatmaps for3 clusters

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [3], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(10,8))
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [3], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical',  fig_size=(10,8))
```

10D. Look at some heatmaps for 4 clusters

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [4], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(10,8))
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [4], [10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(16,12))
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [4], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical',  fig_size=(10,8))
```

10D. Look at some heatmaps for 5 & 6 clusters

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [5], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(10,8))
```

In [ ]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [6], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='kmeans',  fig_size=(10,8))
```

11. Observations & Conclusions

- developed 'full pipeline' functionality
- looked at hyper-parameter optimization
- k-means works really well for 2 clusters
- for 4 clusters, hierarchical is better in sub-grouping the filled and partial filled

12. Next steps:

- move some of the methods here into (new) module for re-use
- also add 'similarity learning' (see unsupervised2 notebook)
- apply the 'full pipeline' on the harder data set

Michael Janus, 29 August 2018 (1:50 am !)

Try two-step pipeline

step 1: filter out black tiles

step 2: cluster remaining tiles

Parametrize:

In [229]:
```
n_clusters_step1 = 3
n_clusters_step2_kmeans = 2
n_clusters_step2_hierarchical = 2

n_patches_x = 12
n_patches_y = 12
```

Feature extract

In [230]:
```
# reset
df = df.drop(columns=['kmeans'])
df2 = None
df3 = None
```

In [231]:
```
imgfiles = import_data(datafolder)
df, feature_names = extract_features(imgfiles, feature_funcs, n_patches_y, n_patches_x)
```

Step 1: filter-out black tiles

In [232]:
```
_ = run_kmeans_pipeline(df, feature_names, n_clusters_step1, standardize=True, use_pca=True )
```

In [233]:
```
#imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(10,8))
```

In [234]:
```
df['kmeans'].value_counts()
```

Out[234]:
```
1    750
0    349
2     53
Name: kmeans, dtype: int64
```

In [235]:
```
# cat_select = 1
# we know for this it's the biggest set
i_max_count = df['kmeans'].value_counts()
cat_select = i_max_count.index[0]
print(cat_select)
```

```
1
```

In [236]:
```
df2 = df[df['kmeans']==cat_select]
df2.head(3)
```

Out[236]:

	filename	s_y	s_x	n_y	n_x	alias	img_std	img_relstd	img_mean	img_skewness	img_kurtosis	img_mode	img_range	kmeans
10	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	10	12	12	img0_0-10	1969.671781	0.204354	9638.507040	-3.281906	9.394410	10293.051270	9527.0	1
11	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	11	12	12	img0_0-11	156.130251	0.015316	10193.827261	-0.433104	1.607858	10186.660645	1817.0	1
23	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	1	11	12	12	img0_1-11	181.149488	0.017557	10317.827261	-0.072174	0.587432	10285.608398	1790.0	1

In [237]:
```
score_kmeans = run_kmeans_pipeline(df2, feature_names, n_clusters_step2_kmeans, standardize=True, use_pca=True )
score_hierarch = run_hierarchical_pipeline(df2, feature_names, n_clusters_step2_hierarchical, standardize=False, use_pca=False )
```

In [238]:
```
df2['kmeans'].value_counts()
```

Out[238]:
```
0    567
1    183
Name: kmeans, dtype: int64
```

In [239]:
```
df2['hierarchical'].value_counts()
```

Out[239]:
```
0    715
1     35
Name: hierarchical, dtype: int64
```

In [240]:
```
df2=df2.rename(columns = {'kmeans':'kmeans2', 'hierarchical':'hierarchical2'})
```

In [241]:
```
df2.head(3)
```

Out[241]:

	filename	s_y	s_x	n_y	n_x	alias	img_std	img_relstd	img_mean	img_skewness	img_kurtosis	img_mode	img_range	kmeans2	hiera
10	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	10	12	12	img0_0-10	1969.671781	0.204354	9638.507040	-3.281906	9.394410	10293.051270	9527.0	1	1
11	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	11	12	12	img0_0-11	156.130251	0.015316	10193.827261	-0.433104	1.607858	10186.660645	1817.0	0	0
23	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	1	11	12	12	img0_1-11	181.149488	0.017557	10317.827261	-0.072174	0.587432	10285.608398	1790.0	0	0

In [242]:
```
df3 = df.merge(df2, 'left')
```

In [243]:
```
df3.head(3)
```

Out[243]:

	filename	s_y	s_x	n_y	n_x	alias	img_std	img_relstd	img_mean	img_skewness	img_kurtosis	img_mode	img_range	kmeans	kmeans2	hier
0	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	0	12	12	img0_0-0	59.727645	0.097910	610.022935	0.214677	0.038107	595.751953	436.0	0	NaN	NaN
1	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	1	12	12	img0_0-1	59.615076	0.094367	631.737407	0.245826	0.158095	602.675293	489.0	0	NaN	NaN
2	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	2	12	12	img0_0-2	58.523271	0.090539	646.383510	0.257270	0.134783	651.507324	431.0	0	NaN	NaN

In [244]:
```
df3['kmeans2'].fillna(value=-1, inplace=True)
```

In [245]:
```
df3['hierarchical2'].fillna(value=-1, inplace=True)
```

In [246]:
```
df3.head(3)
```

Out[246]:

	filename	s_y	s_x	n_y	n_x	alias	img_std	img_relstd	img_mean	img_skewness	img_kurtosis	img_mode	img_range	kmeans	kmeans2	hier
0	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	0	12	12	img0_0-0	59.727645	0.097910	610.022935	0.214677	0.038107	595.751953	436.0	0	-1.0	-1.0
1	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	1	12	12	img0_0-1	59.615076	0.094367	631.737407	0.245826	0.158095	602.675293	489.0	0	-1.0	-1.0
2	..\data\Crystals_Apr_12\Tileset6_Subset_Blacks...	0	2	12	12	img0_0-2	58.523271	0.090539	646.383510	0.257270	0.134783	651.507324	431.0	0	-1.0	-1.0

In [247]:
```
df3['heats']=df3['kmeans2']+1
```

In [248]:
```
df3['heats'].value_counts()
```

Out[248]:
```
1.0    567
0.0    402
2.0    183
Name: heats, dtype: int64
```

In [249]:
```
df3['heats'].replace({1:0}, inplace=True)
df3['heats'].value_counts()
```

Out[249]:
```
0.0    969
2.0    183
Name: heats, dtype: int64
```

In [250]:
```
df3['heats2']=df3['hierarchical2']+1
```

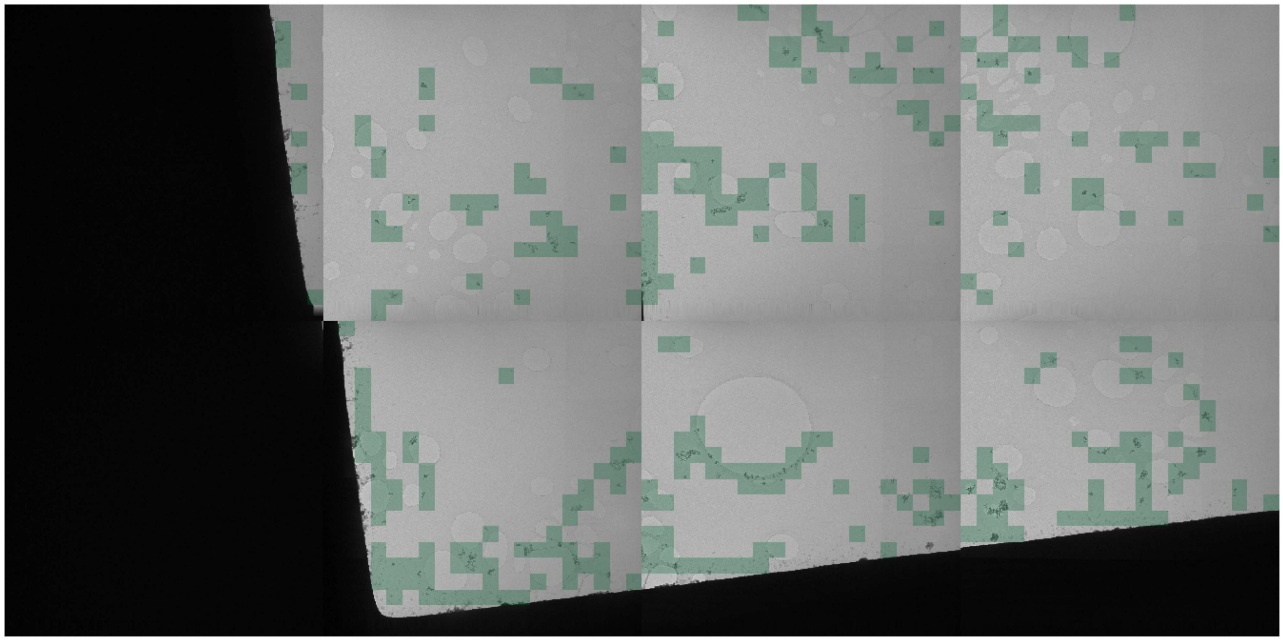
In [251]:
```
df3['heats2'].value_counts()
```

Out[251]:
```
1.0    715
0.0    402
2.0     35
Name: heats2, dtype: int64
```

In [252]:
```
#swap_clusters(df3, 'heats', 1,2)
```

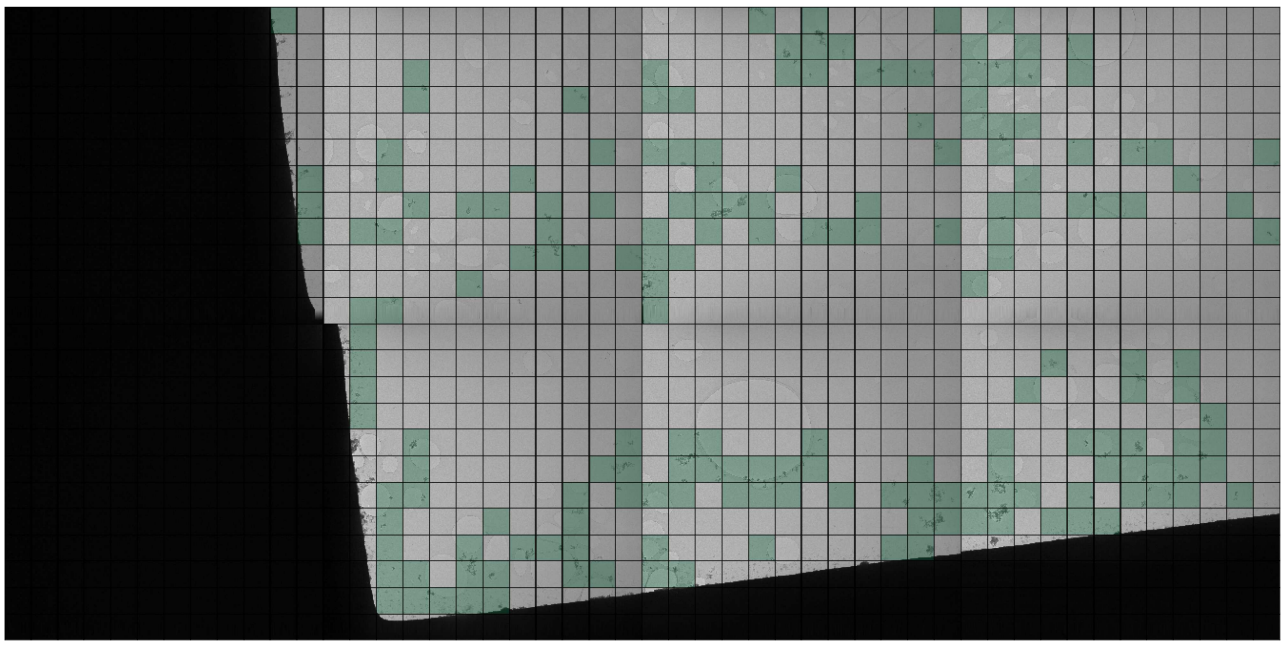
In [186]:
```
imgutils.show_large_heatmap(df3, 'heats', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12), heatdependent_opacity=True, no_borders=True)
```

Heats from: heats



In [253]:
```
imgutils.show_large_heatmap(df3, 'heats', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12), heatdependent_opacity=True)
```

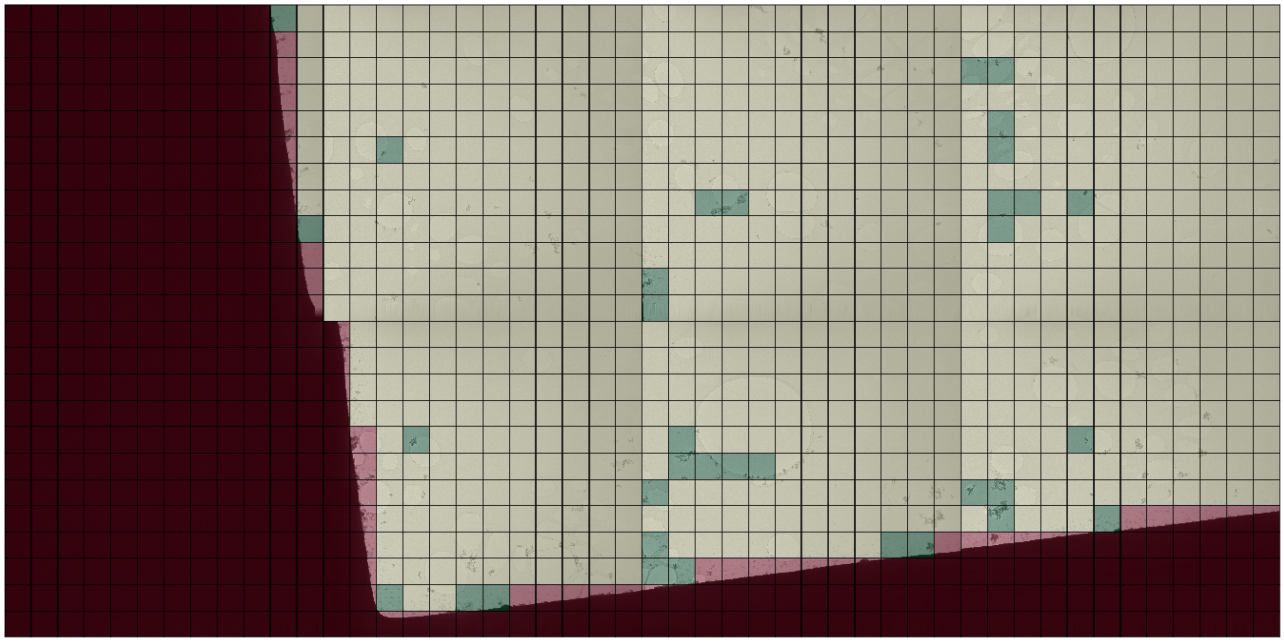
Heats from: heats



In [ ]:
```
swap_clusters(df3, 'heats', 1,2)
```

In [153]:
```
imgutils.show_large_heatmap(df3, 'heats2', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12))
```

Heats from: heats2



ANother subset

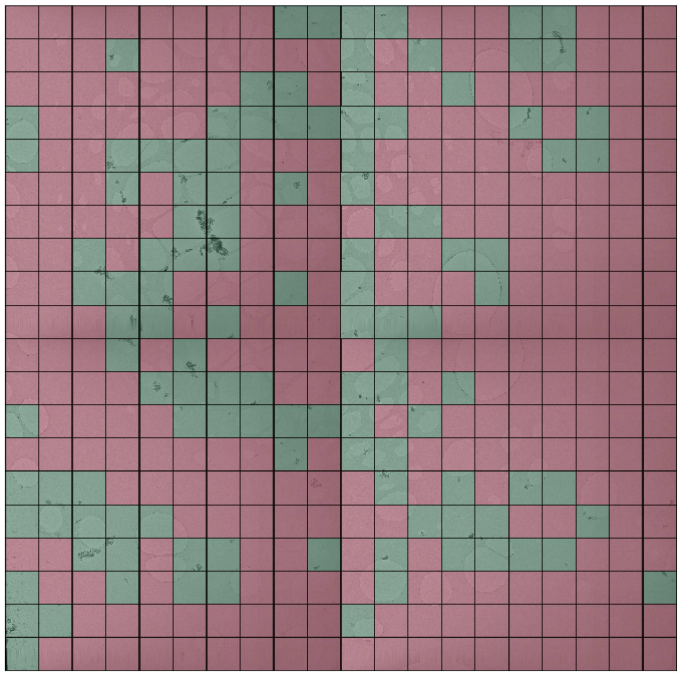
In [272]:
```
datafolder2 = '../data/Crystals_Apr_12/Tileset6_Subset_NoBlack_1k'
n_tiles2_x = 2  # mostly for visualization
n_tiles2_y = 2
```

In [273]:
```
imgfiles2 = imgutils.scanimgdir(datafolder2, '.tif')['filename'].tolist()
```

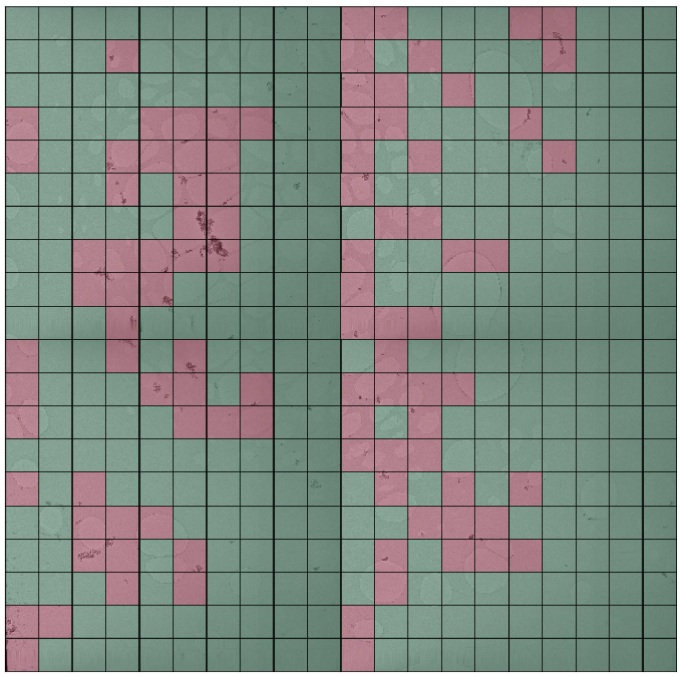
In [274]:
```
df = run_fullpipeline(datafolder2, n_tiles2_y, n_tiles2_x, 10, 10, feature_funcs, 2, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 253.8806458337664
Score hierarchical: 562.7385664608233
Visualizing...
```

Heats from: kmeans

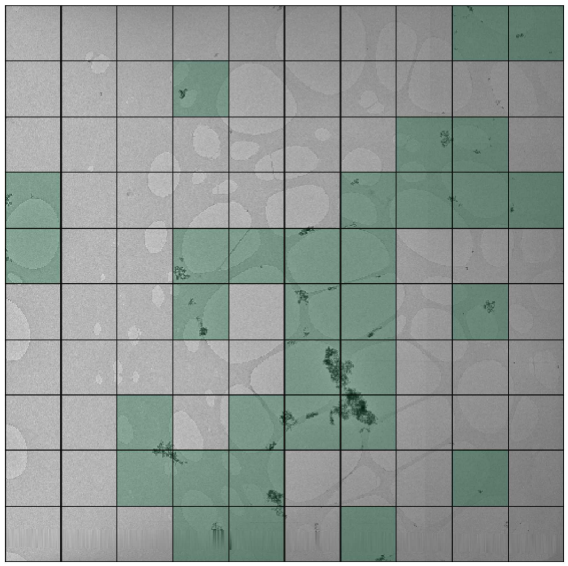


Heats from: hierarchical

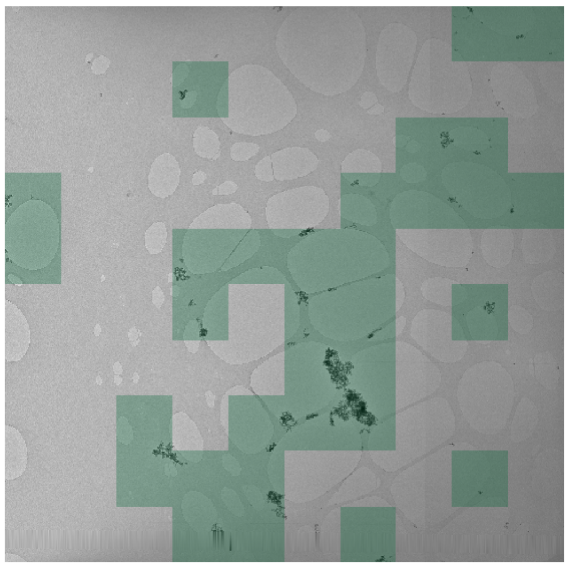


In [260]:
```
imgutils.show_large_heatmap(df, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=False, heatdependent_opacity=True)
imgutils.show_large_heatmap(df, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=True, heatdependent_opacity=True)
```

Heats from: kmeans



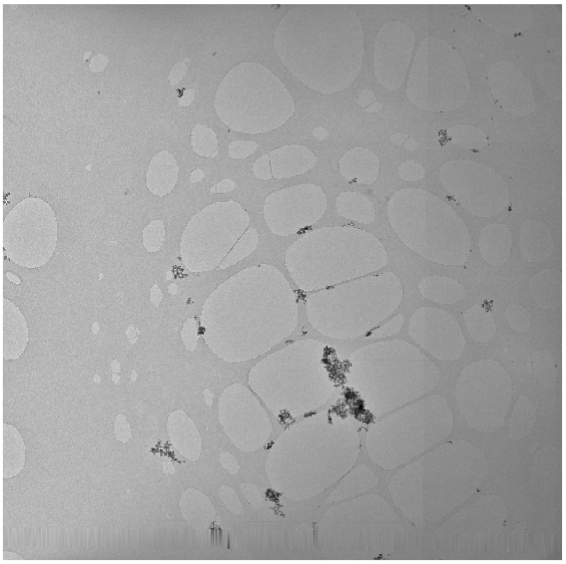
Heats from: kmeans



In [277]:
```
df['dummy']=0
```

In [278]:
```
imgutils.show_large_heatmap(df, 'dummy', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=True, heatdependent_opacity=True)
```

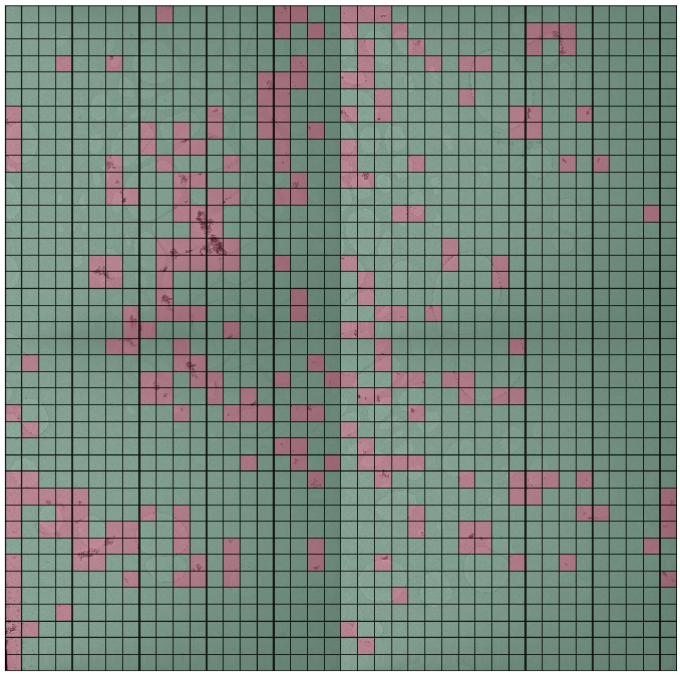
Heats from: dummy



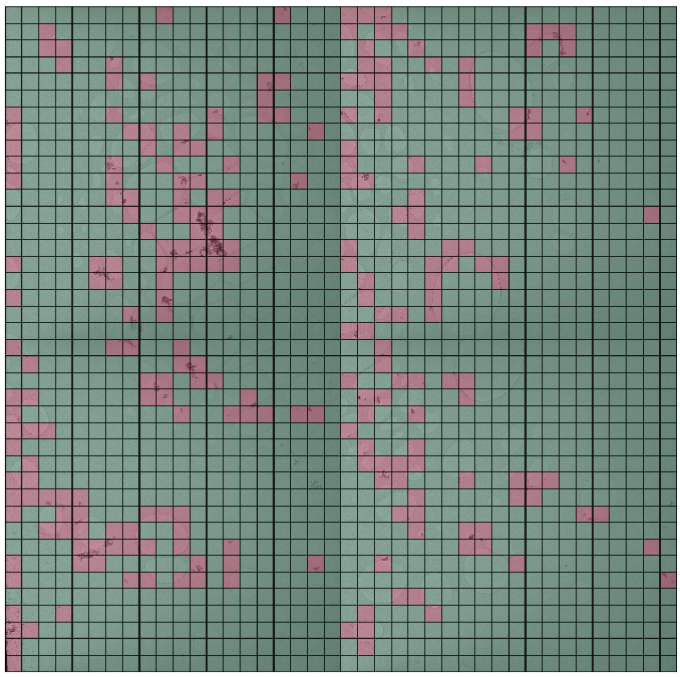
In [275]:
```
df2 = run_fullpipeline(datafolder2, n_tiles2_y, n_tiles2_x, 20, 20, feature_funcs, 2, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 1007.888966651257
Score hierarchical: 1864.143277978098
Visualizing...
```

Heats from: kmeans

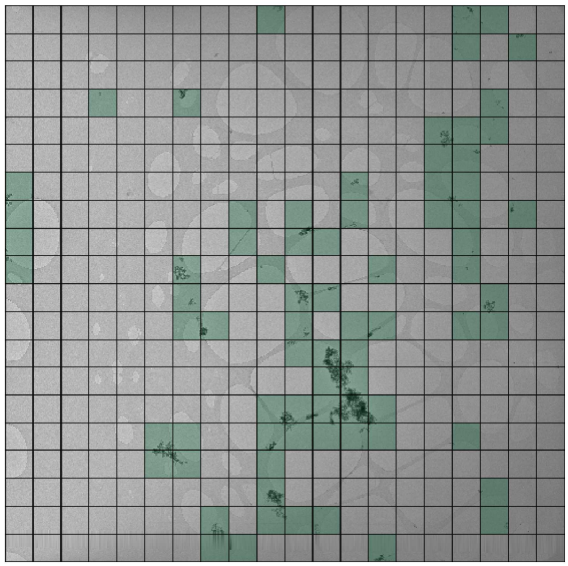


Heats from: hierarchical

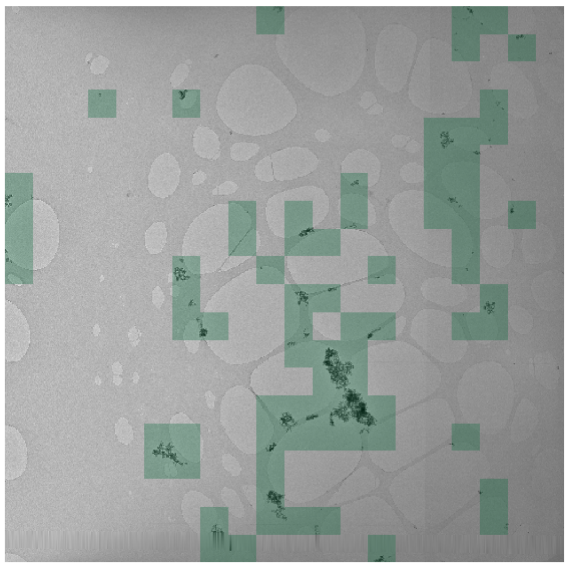


In [264]:
```
imgutils.show_large_heatmap(df2, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=False, heatdependent_opacity=True)
imgutils.show_large_heatmap(df2, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=True, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans

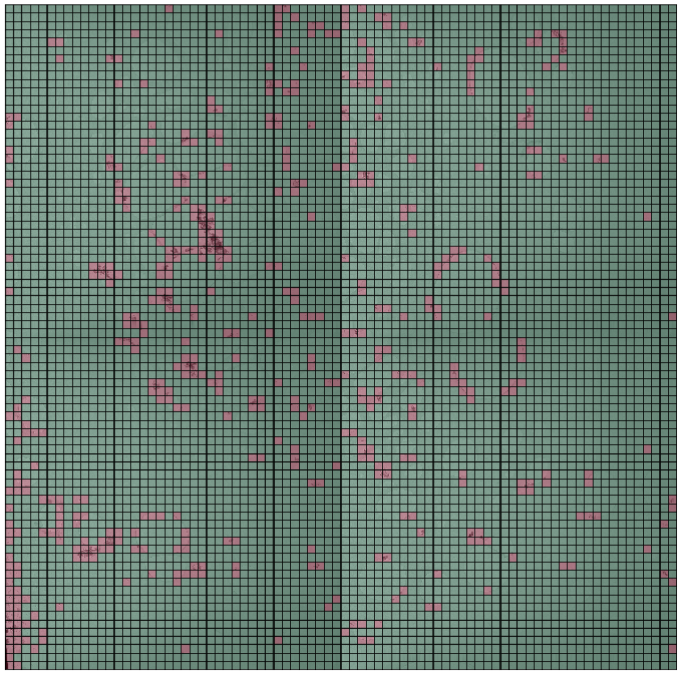


run with very fine grid

In [265]:
```
df4 = run_fullpipeline_kmeans(datafolder2, n_tiles2_y, n_tiles2_x, 40, 40, feature_funcs, 2, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 3472.5680761978
Visualizing...
```

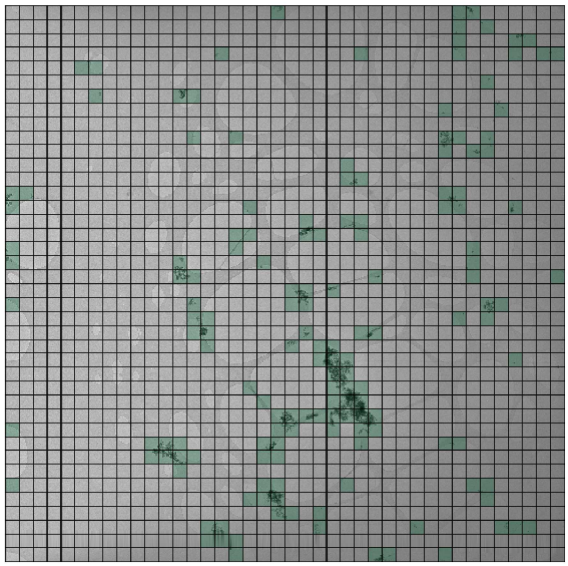
Heats from: kmeans



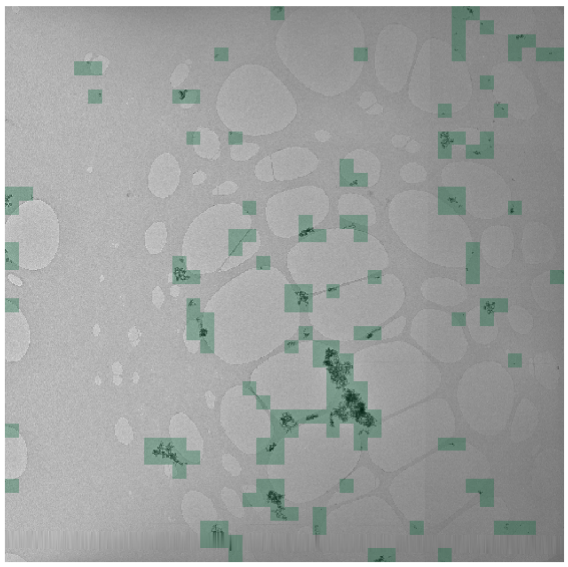
In [267]:
```
swap_clusters(df4, 'kmeans',0,1)
```

In [268]:
```
imgutils.show_large_heatmap(df4, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=False, heatdependent_opacity=True)
imgutils.show_large_heatmap(df4, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=True, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans



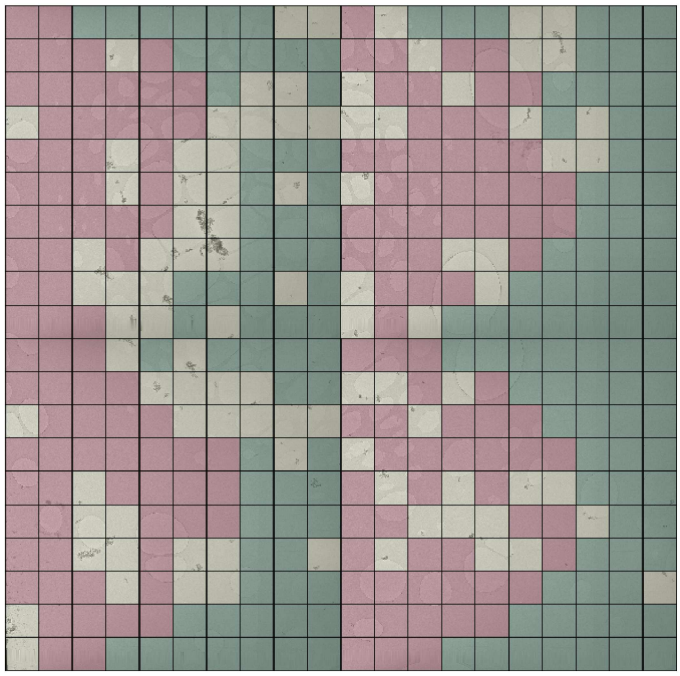
In [ ]:
```
imgutils.show_large_heatmap(df4, 'kmeans', [imgfiles2[0]], n_rows=1, n_cols=1, fig_size=(12,10), no_borders=False, heatdependent_opacity=True)
```

3 clusters:

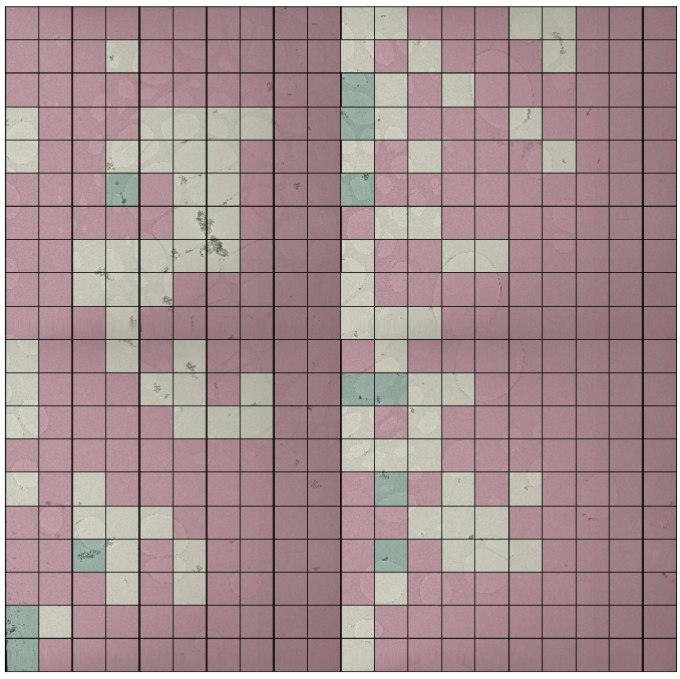
In [104]:
```
df = run_fullpipeline(datafolder2, n_tiles2_y, n_tiles2_x, 10, 10, feature_funcs, 3, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 271.31570954292135
Score hierarchical: 331.34447740342443
Visualizing...
```

Heats from: kmeans



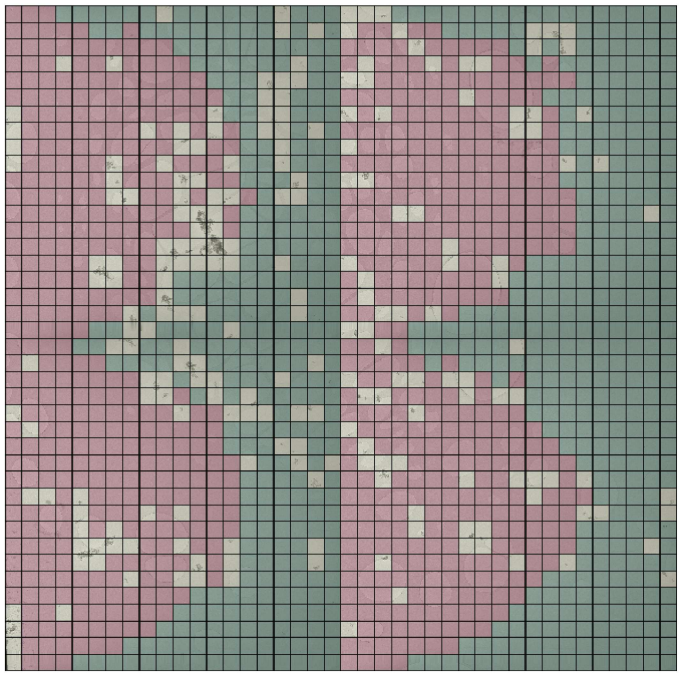
Heats from: hierarchical



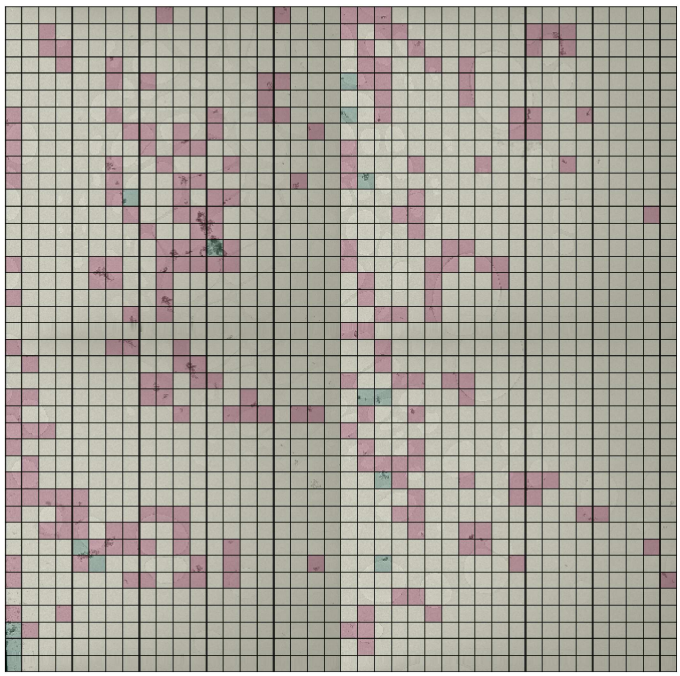
In [105]:
```
df = run_fullpipeline(datafolder2, n_tiles2_y, n_tiles2_x, 20, 20, feature_funcs, 3, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 1195.997631503202
Score hierarchical: 1063.9415055648
Visualizing...
```

Heats from: kmeans



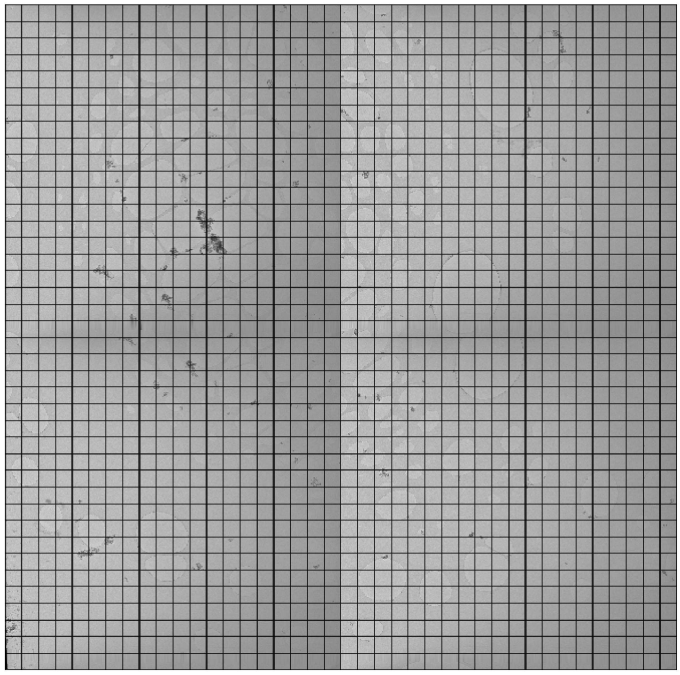
Heats from: hierarchical



In [126]:
```
df['dummy'] = 0
```

In [127]:
```
imgfiles = df['filename'].unique()
imgutils.show_large_heatmap(df, 'dummy', imgfiles, n_rows=n_tiles2_y, n_cols=n_tiles2_x, fig_size=(16,12))
```

Heats from: dummy



In [128]:
```
# try the feature-similarity on this set
import sklearn.preprocessing as skpreproc
```

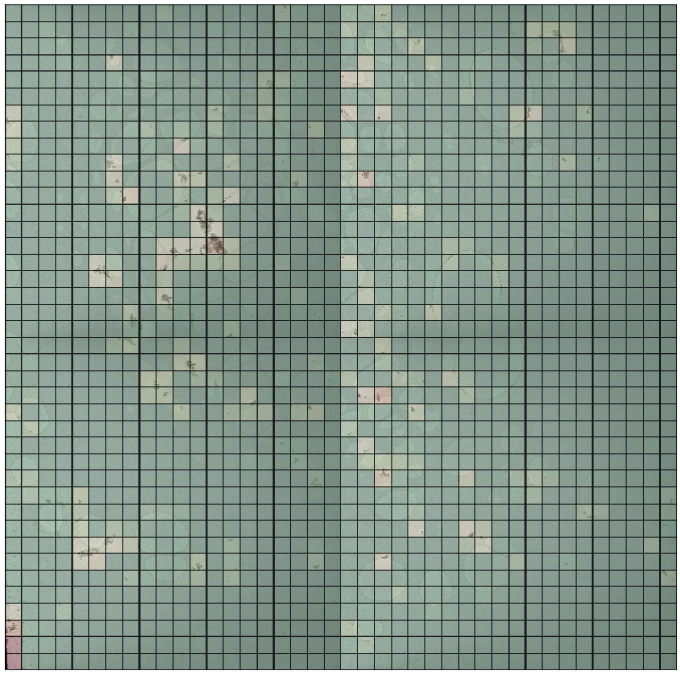
In [129]:
```
X = df.loc[:,feature_names]
```

In [130]:
```
X_norm = skpreproc.normalize(X) # the orignal features, but normalized per sample
```

In [131]:
```
# pick any as the reference
ref_vect = X_norm[11]
df['simscore'] = X_norm.dot(ref_vect)
```

In [132]:
```
imgutils.show_large_heatmap(df, 'simscore', imgfiles, n_rows=n_tiles2_y, n_cols=n_tiles2_x, fig_size=(16,12))
```

Heats from: simscore



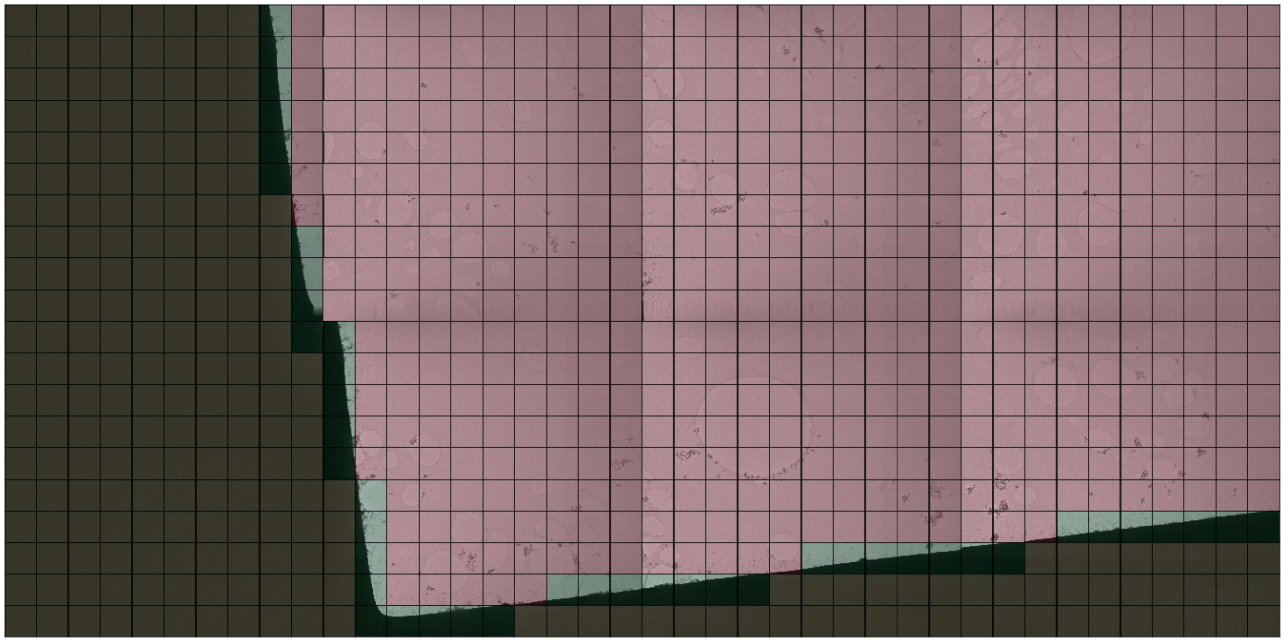
No machine learning, just the feature vector!

In [ ]:
```
# try this out with the black stuff
```

In [133]:
```
df = run_fullpipeline_kmeans(datafolder, n_tiles_y, n_tiles_x, 10, 10, feature_funcs, 3, fig_size=(16,12), return_df=True)
```

```
Working...
['img_std', 'img_relstd', 'img_mean', 'img_skewness', 'img_kurtosis', 'img_mode', 'img_range']
Results:
Score k-means: 729.4224696533237
Visualizing...
```

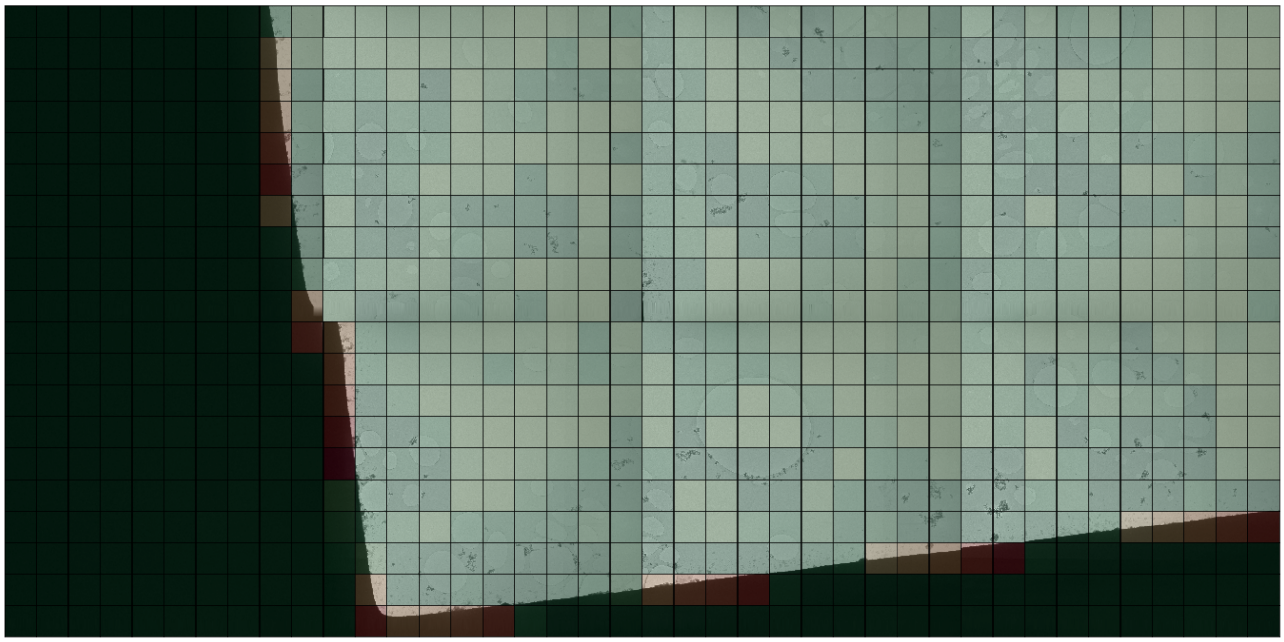
Heats from: kmeans



In [136]:
```
X = df.loc[:,feature_names]
X_norm = skpreproc.normalize(X) # the orignal features, but normalized per sample
# pick any as the reference
ref_vect = X_norm[11]
df['simscore'] = X_norm.dot(ref_vect)

imgfiles = imgutils.scanimgdir(datafolder, '.tif')['filename'].tolist()
imgutils.show_large_heatmap(df, 'simscore', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12))
```

Heats from: simscore



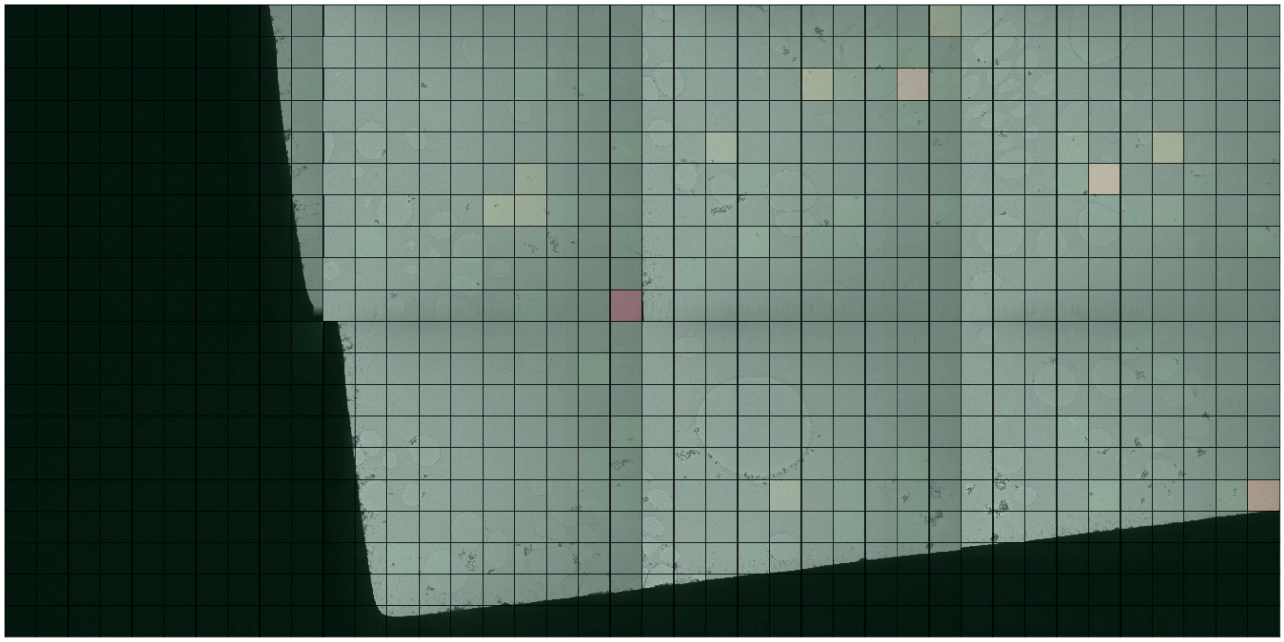
In [140]:
```
imgutils.stat_names(try_funcs)
```

Out[140]:
```
['img_std', 'img_relstd', 'img_skewness', 'img_kurtosis']
```

In [141]:
```
# once more without mode and mean:
other_feat = imgutils.stat_names(try_funcs)
X = df.loc[:,other_feat]
X_norm = skpreproc.normalize(X) # the orignal features, but normalized per sample
# pick any as the reference
ref_vect = X_norm[11]
df['simscore'] = X_norm.dot(ref_vect)

imgfiles = imgutils.scanimgdir(datafolder, '.tif')['filename'].tolist()
imgutils.show_large_heatmap(df, 'simscore', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12))
```

Heats from: simscore



No, feat-vector not working here either (I guess again because of the black)

Extra Visualizations for report (Nov 2018)

In [1]:
```
def change_clusternums(df, columnname, oldnew_dict):
    df[columnname].replace(oldnew_dict, inplace=True)

def swap_clusters(df, columnname, clust1, clust2):
    oldnew_dict = { clust1: clust2, clust2: clust1}
    df[columnname].replace(oldnew_dict, inplace=True)

s = (12,12)
```

In [271]:
```
# this is a set of 2 x 3 images covering a larger area (a so called 'tile set')
imgutils.showimgset(imgfiles, 2,4, fig_size=(12, 8), relspacing=(0.05,0.05))
```