

Full Pipeline (on Tileset7) - Aug 2017

Created: 21 Aug 2018
Last update: 4 Nov 2018

Goal: Combine the relevant steps from data import to unsupervised learning

Many functions have gradually been developed in the prior notebooks (and added to 'imgutils'). In this notebook, the steps will be combined without all the intermediate analysis.

1. Imports

```
In [183]: # 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.decomposition import PCA, TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.metrics import silhouette_score

import imgutils

In [184]: # Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)

Out[184]: <module 'imgutils' from 'C:\\JADS\\SW\\Grad Proj\\sources\\imgutils.py'>
```

2. Data Definitions & Feature Specification

```
In [304]: # Data:
datafolder = '../data/Crystals_Apr_12/Tileset7'
n_tiles_x = 3 # 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]
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 [186]: # image import:
print("Scanning for images in '{}'...".format(datafolder))
df_imgfiles = imgutils.scandir(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/Tileset7'...
# of images: 6

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
# slices extracted: 96
```

4. Machine Learning Pipeline

Hyper parameters

```
In [187]: # data hyper-parameters
default_n_clusters = 3

# algorithm hyper-parameters:
kmeans_n_init = 20
```

```
In [188]: 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 [189]: 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("Inclustering Scores:")
    print("K-means: ", score_kmeans)
    print("Hierarchical: ", score_hier)
```

```
In [190]: run_ml_pipelines2(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done
```

```
Clustering Scores:
K-means:  0.38346805711017223
Hierarchical:  0.7100243618056425
```

Why do these scores deviate from previous notebook? Is there something going on with the sklearn pipeline or StandardScaler?

5. Investigated scoring issue in other notebook

(see realxtals1-pipeline_scoring_issues)

Nothing was wrong with the pipeline or original step-by-step implementation. It was caused by different basis for the score calculation (pipeline impl. is using the original data while the step-by-step is using the normalized data for looking at the 'internal clustering score').

As such, turning normalization off (in the pipeline) gives higher scores. This does not mean the outcome is better, that needs visual inspection.

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 [191]: 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)

    return score, y

In [27]: 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("Inclustering Scores:")
    print("K-means: ", score_kmeans)
    print("Hierarchical: ", score_hier)
```

```
In [11]: run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)

Executing clustering pipelines...
Done
```

```
Clustering Scores:
K-means:  0.5251916981567152
Hierarchical:  0.5543189694833234
```

5. Visualize results

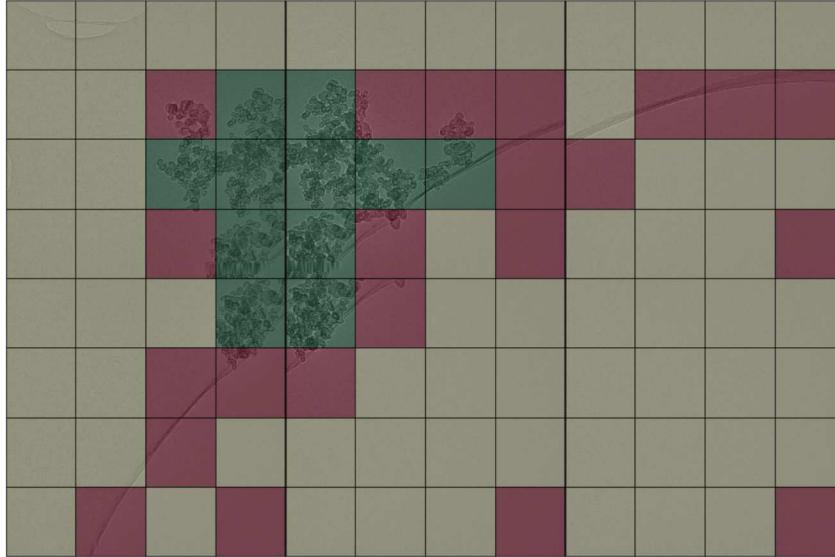
```
In [12]: run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done
```

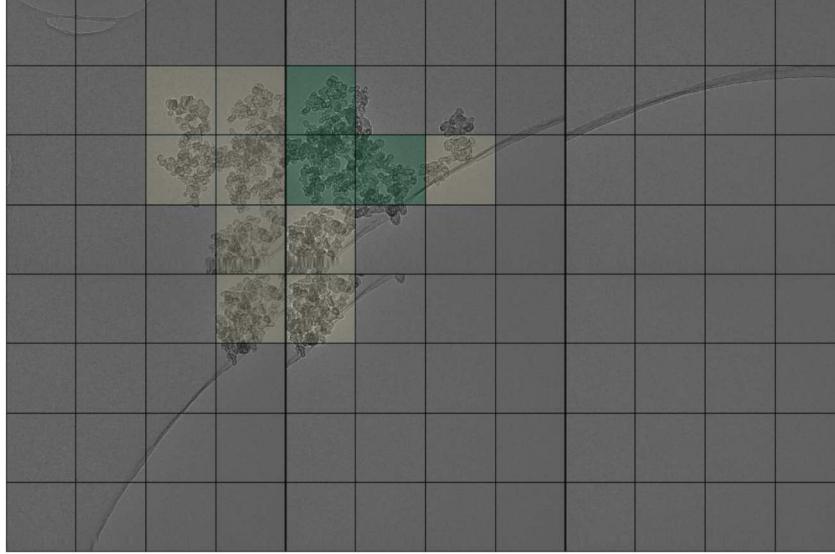
```
Clustering Scores:
K-means: 0.5251916981567152
Hierarchical: 0.5543189694833234
```

```
In [192]: s = (12,10)
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, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: hierarchical



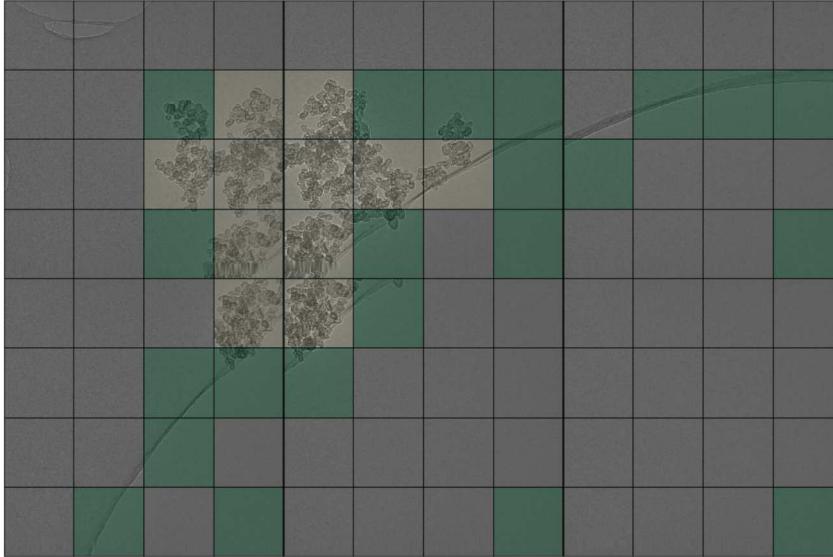
Run it again without PCA and/pr normalization compare results

```
In [193]: 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, heatdependent_opacity=True)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s, heatdependent_opacity=True)

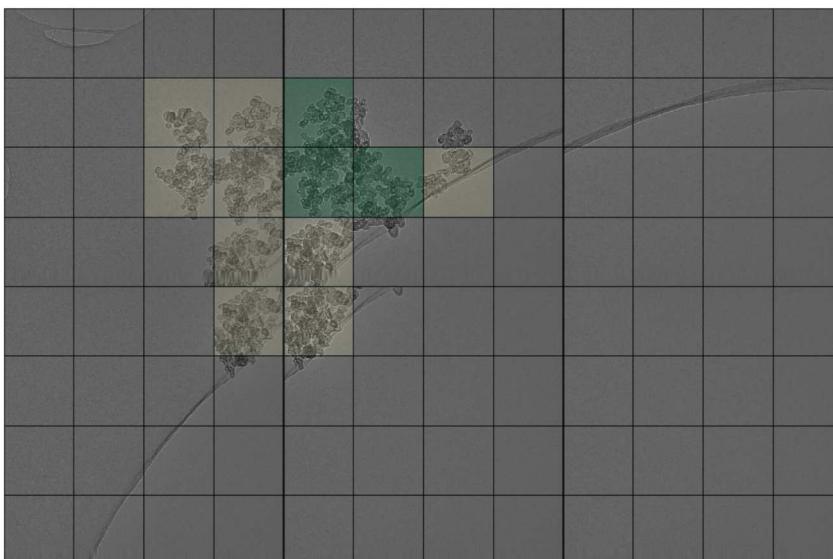
Executing clustering pipelines...
Done
```

Clustering Scores:
K-means: 0.5251916981567151
Hierarchical: 0.5543189694833234

Heats from: kmeans



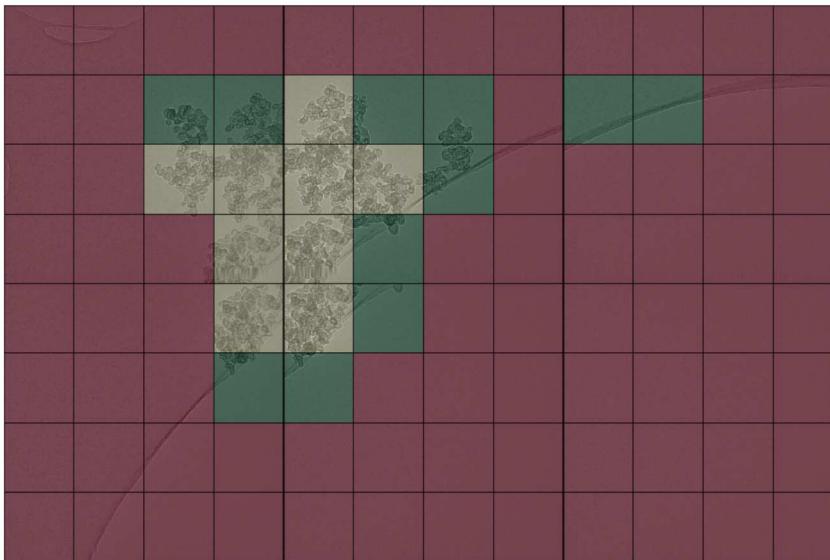
Heats from: hierarchical



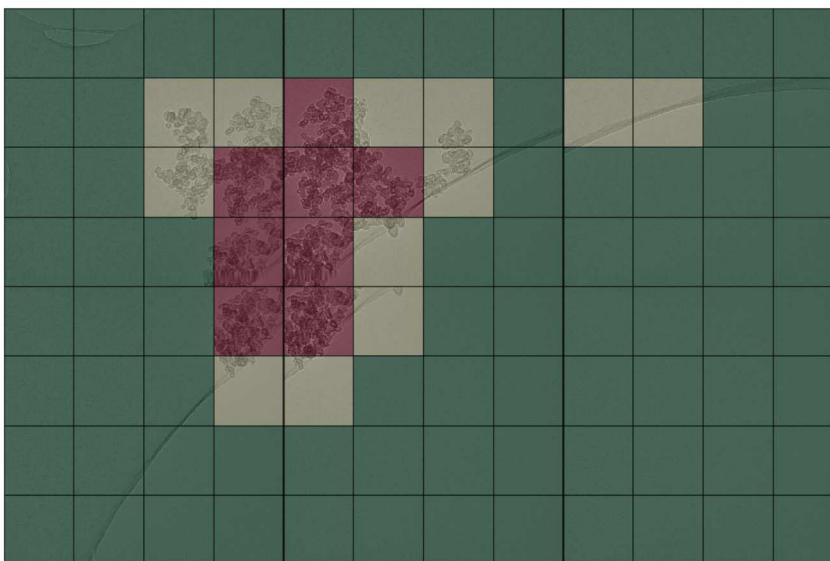
```
In [194]: run_ml_pipelines(df, feature_names, default_n_clusters, standardize=False, use_pca=False)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
Executing clustering pipelines...
Done
```

Clustering Scores:
K-means: 0.600017787779939
Hierarchical: 0.6138055241918071

Heats from: kmeans



Heats from: hierarchical



On this dataset, hierarchical works slightly better without normalization (!?)!

Hence, the method that runs two pipelines is less useful as they need different parameterization. So let's make two methods below that run the full pipeline including slicing

6. Combine import and pipeline:

```
In [195]: def import_data(imagefolder):
    df_imgfiles = imgutils.scanimgdir(datafolder, '.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 [196]: 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 [197]: def run_fullpipeline(imagefolder, n_image_rows, n_image_cols,
                           n_grid_rows, n_grid_cols, feature_funcs, n_clusters,
                           fig_size = (10,8), 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)
    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...')
    s = (8,6)
    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 []:

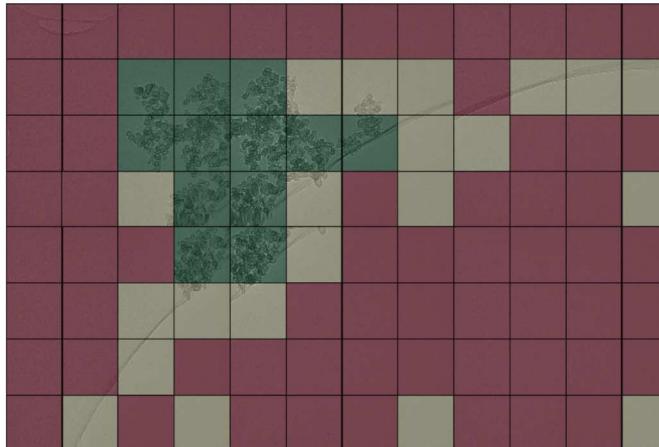
7. Try it out with different combinations of slices

4x4 - 3 clusters

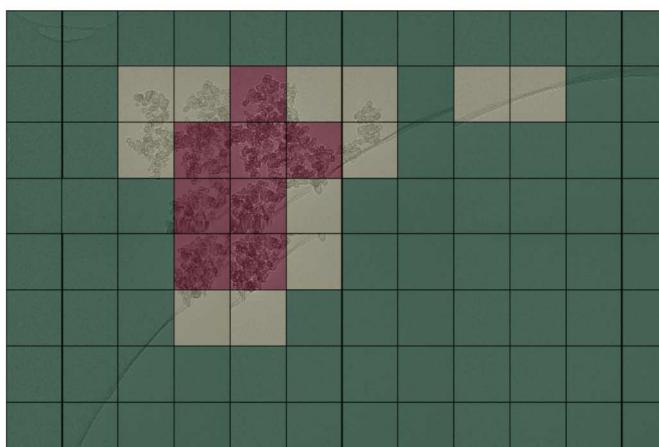
```
In [198]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 3)

Working...
Results:
Score k-means: 0.5272951661092292
Score hierarchical: 0.6138055241918071
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical



6x6, 3 clusters

```
In [199]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 3)
```

Working...

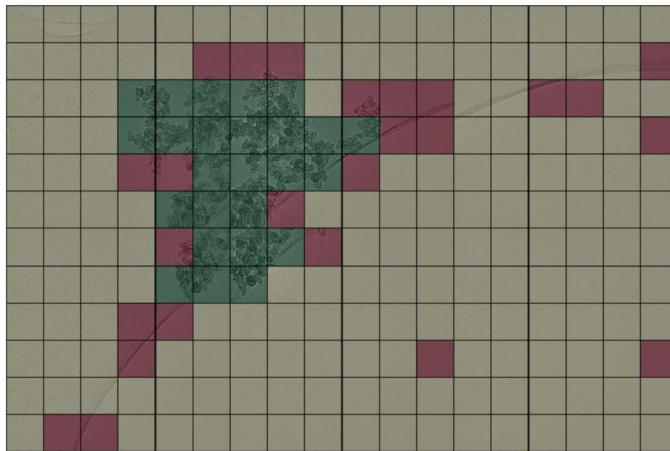
Results:

Score k-means: 0.6020413797895592

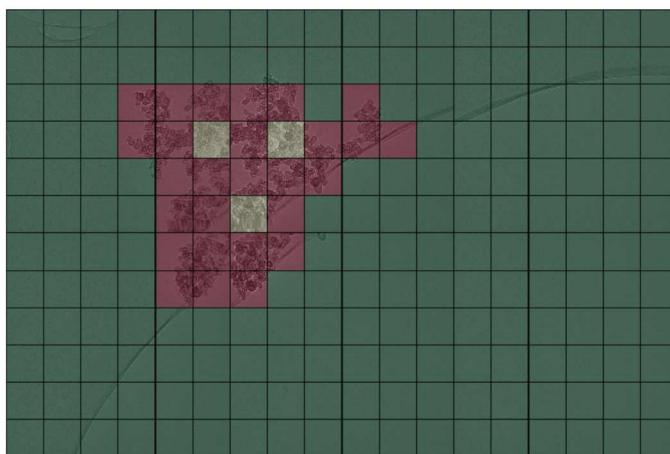
Score hierarchical: 0.7674936226598582

Visualizing...

Heats from: kmeans



Heats from: hierarchical

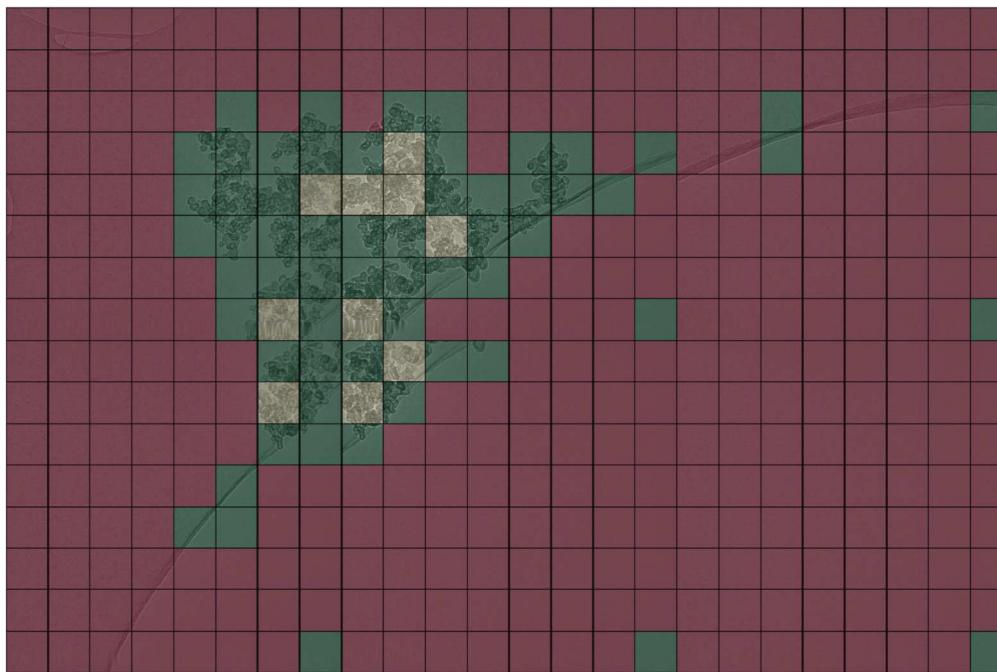


8x8, 3 clusters

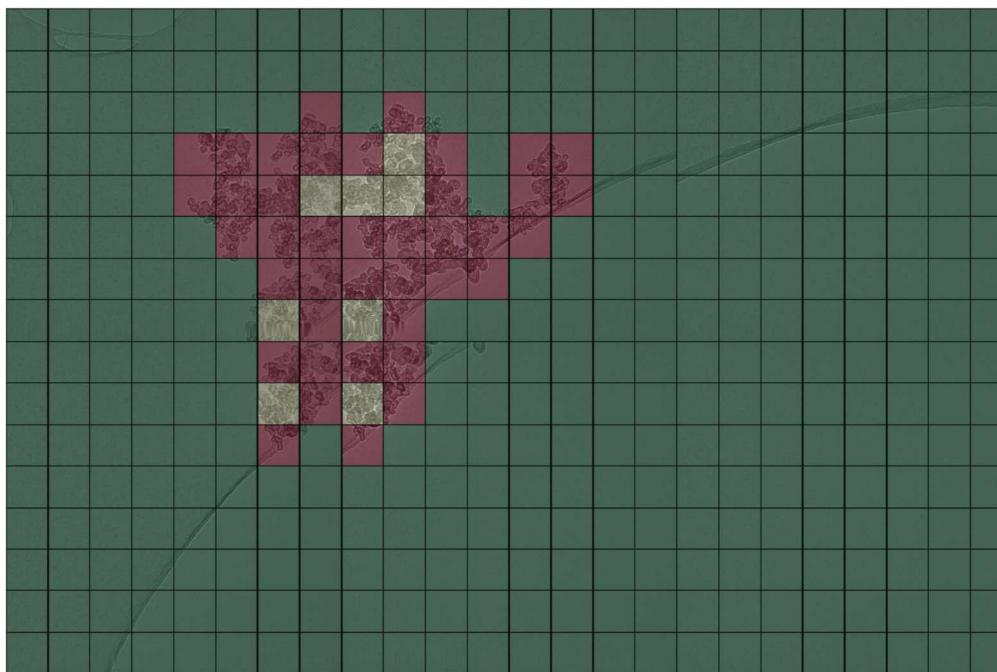
```
In [200]: _ = run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 3, fig_size=(16,12))
```

Working...
Results:
Score k-means: 0.6623808566364214
Score hierarchical: 0.7699361876771397
Visualizing...

Heats from: kmeans



Heats from: hierarchical



8. Try it out with different number of clusters

2 clusters (4x4 , 6x6, 8x8)

```
In [201]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 2)
```

Working...

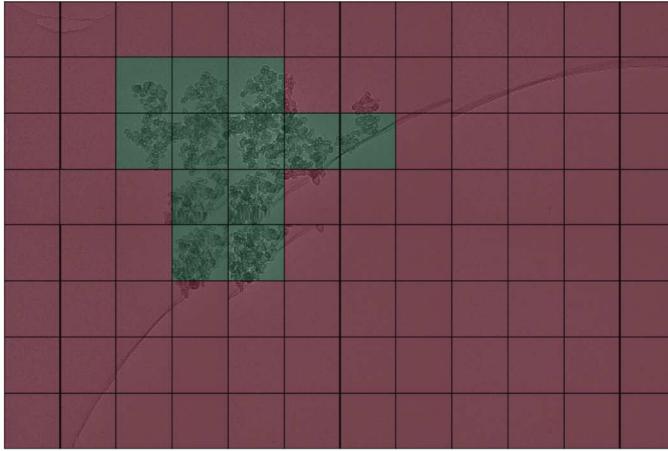
Results:

Score k-means: 0.5886428860898483

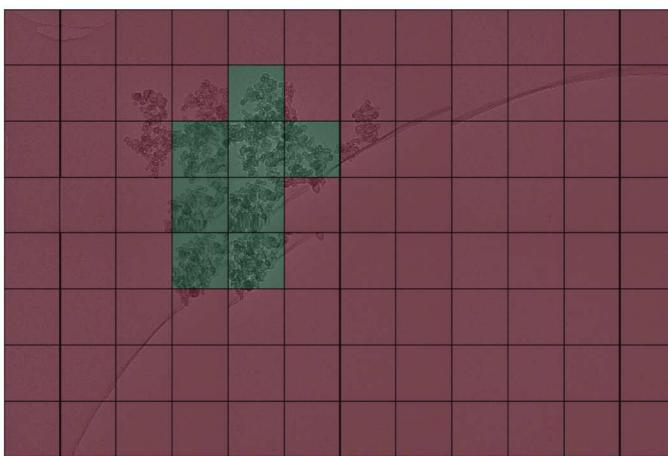
Score hierarchical: 0.7507303799260893

Visualizing...

Heats from: kmeans



Heats from: hierarchical



```
In [202]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 2)
```

Working...

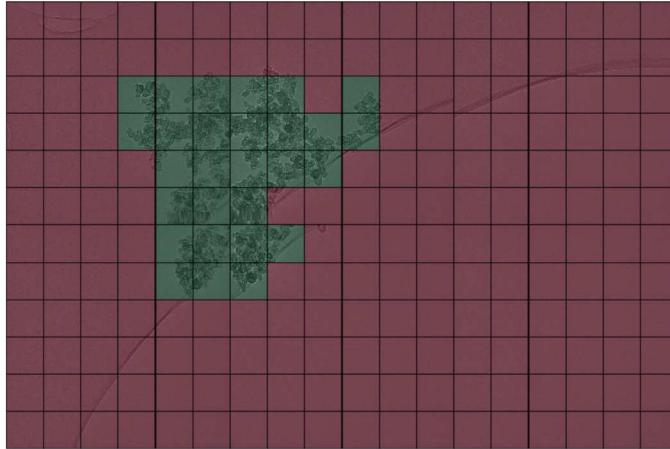
Results:

Score k-means: 0.633254261385902

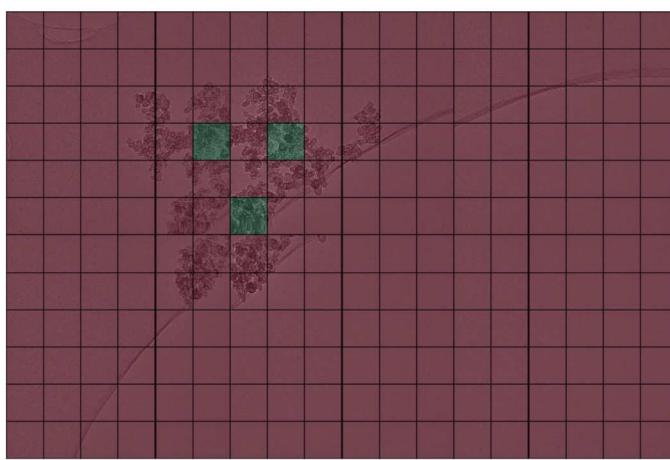
Score hierarchical: 0.7959516863373638

Visualizing...

Heats from: kmeans



Heats from: hierarchical



```
In [203]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 2)
```

Working...

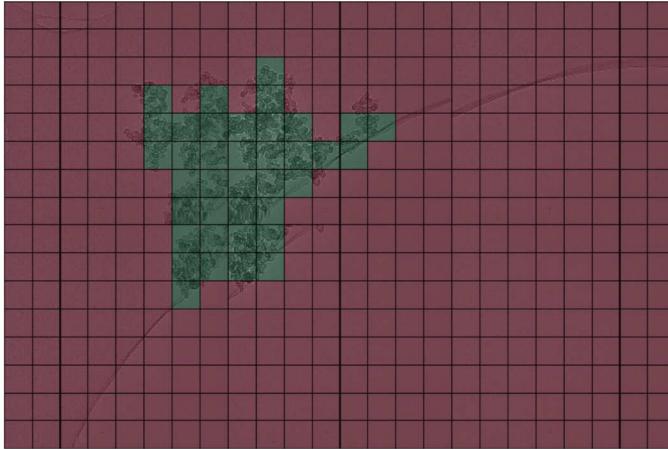
Results:

Score k-means: 0.6769843219186112

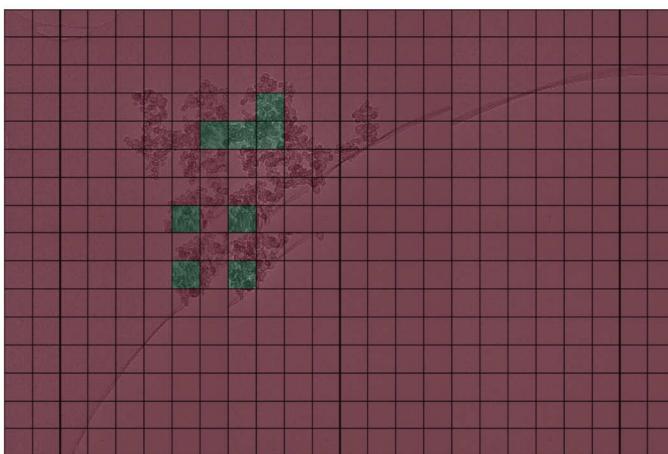
Score hierarchical: 0.8396991318389545

Visualizing...

Heats from: kmeans



Heats from: hierarchical

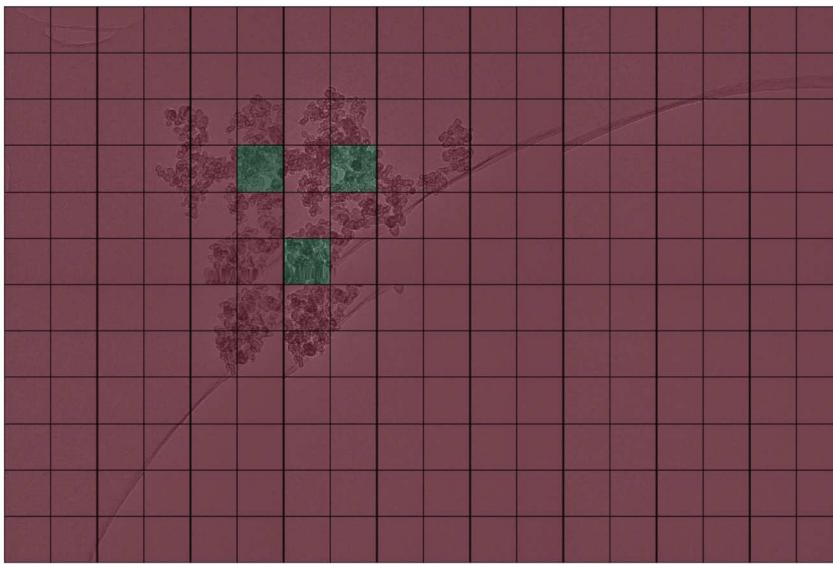


Look again at hierarchical with scaling and pca on:

```
In [204]: imgfiles = import_data(datafolder)
df_temp, feature_names = extract_features(imgfiles, feature_funcs, 6, 6)
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: 0.7360355168056918

Heats from: hierarchical



Without scaling & pca: 0.7959516863373638

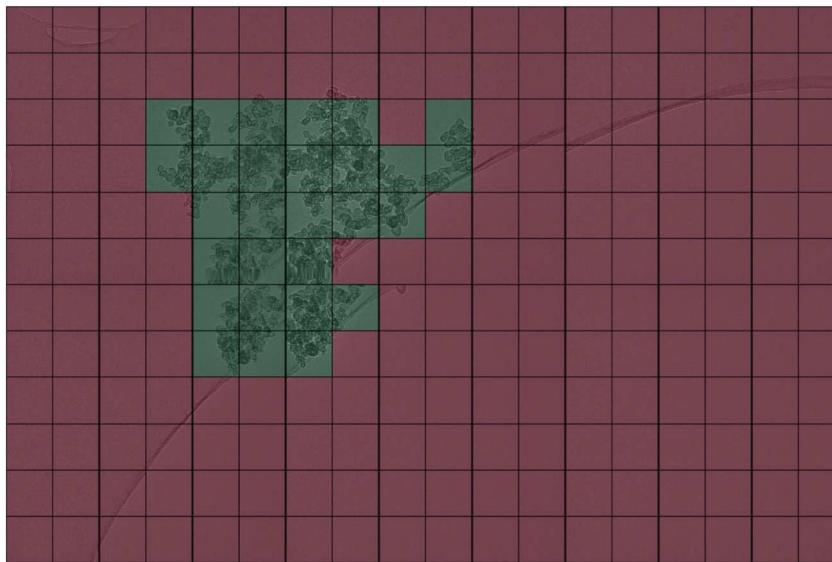
Heats from: hierarchical



```
In [205]: 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)
```

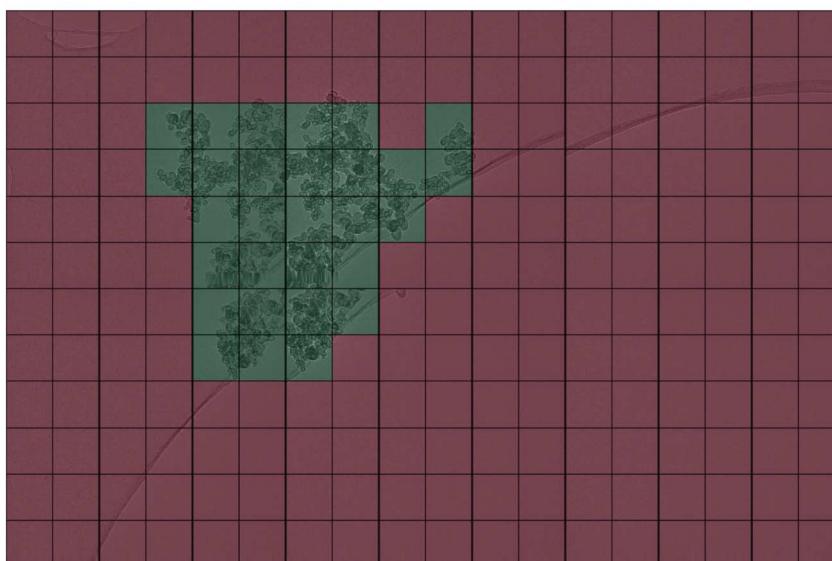
With scaling & pca: 0.633254261385902

Heats from: kmeans



Without scaling & pca: 0.7638653726133209

Heats from: kmeans



Results with and without scaling / pca are pretty close actually.

4 clusters (4x4, 6x6, 8x8)

```
In [206]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 4)
```

Working...

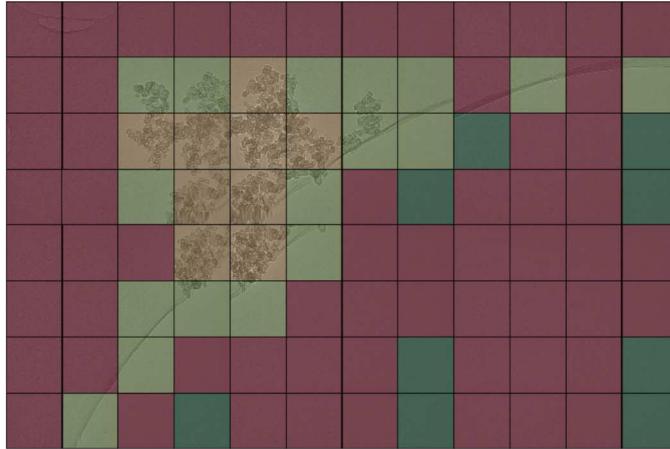
Results:

Score k-means: 0.4892202942275933

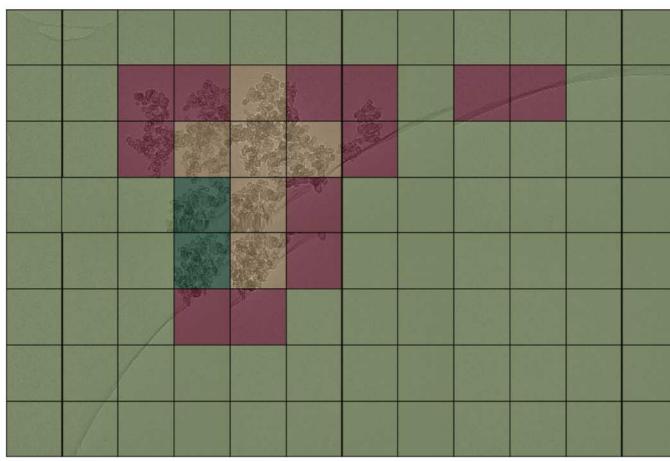
Score hierarchical: 0.6079930287147731

Visualizing...

Heats from: kmeans



Heats from: hierarchical



```
In [207]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 4)
```

Working...

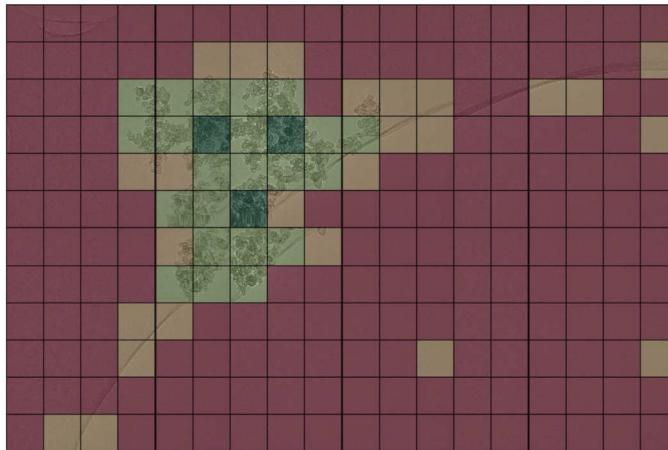
Results:

Score k-means: 0.621537336514108

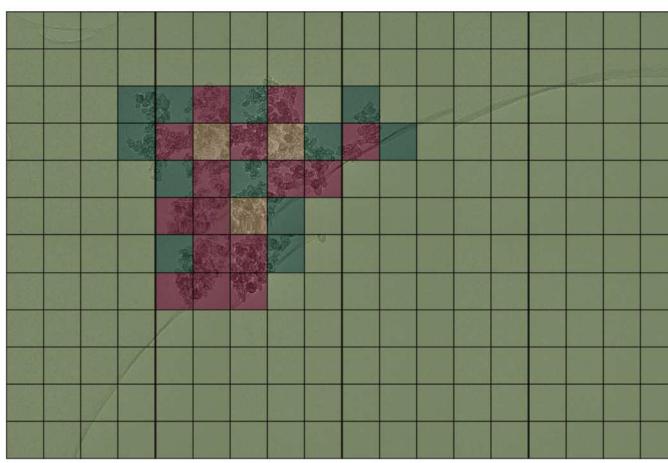
Score hierarchical: 0.7080629222772171

Visualizing...

Heats from: kmeans



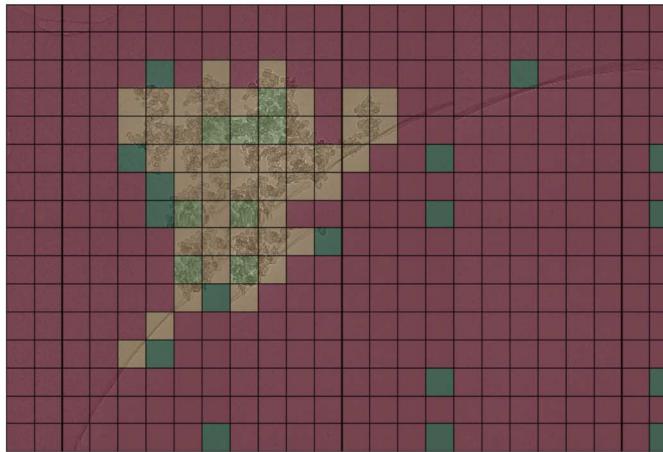
Heats from: hierarchical



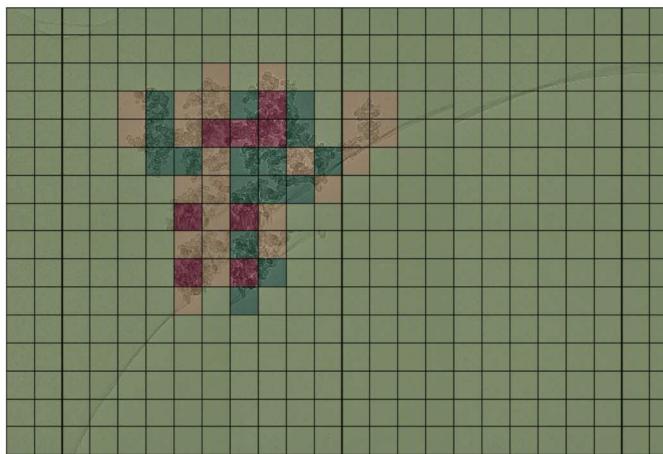
```
In [208]: run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 4)
```

```
Working...
Results:
Score k-means: 0.6849544008660838
Score hierarchical: 0.7220216155001514
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical



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

9. (Grid) Search for hyper-parameter optimizations

```
In [425]: imgfiles = import_data(datafolder)
df, feature_names = extract_features(imgfiles, feature_funcs, 10, 10)
```

9A. Number of clusters

```
In [426]: 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)

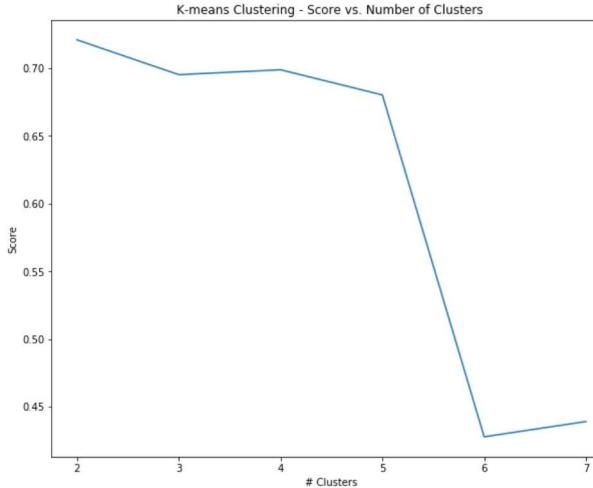
    fig = plt.figure(figsize=(10,8))
    plt.plot(result[:,0], result[:,1])
    plt.title('K-means Clustering - Score vs. Number of Clusters ')
    plt.xlabel('# Clusters')
    plt.ylabel('Score')
    plt.show()
```

```
In [427]: 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)

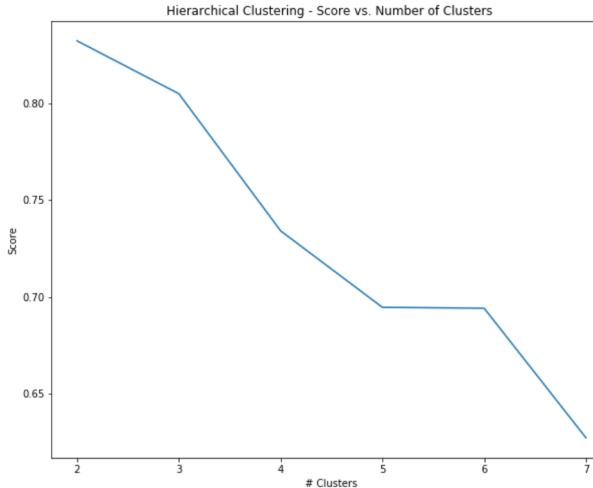
    for n in range(n_clusters_start, n_clusters_end+1):
        score = run_hierarchical_pipeline(df, feature_names, n, standardize=False, use_pca=False )
        result = np.append(result, np.array([[n, score]]), axis=0)

    fig = plt.figure(figsize=(10,8))
    plt.plot(result[:,0], result[:,1])
    plt.title('Hierarchical Clustering - Score vs. Number of Clusters ')
    plt.xlabel('# Clusters')
    plt.ylabel('Score')
    plt.show()
```

```
In [428]: optimize_kmeans_nclusters(df, feature_names, 2, 7)
```



```
In [429]: optimize_hierarchical_nclusters(df, feature_names, 2, 7)
```

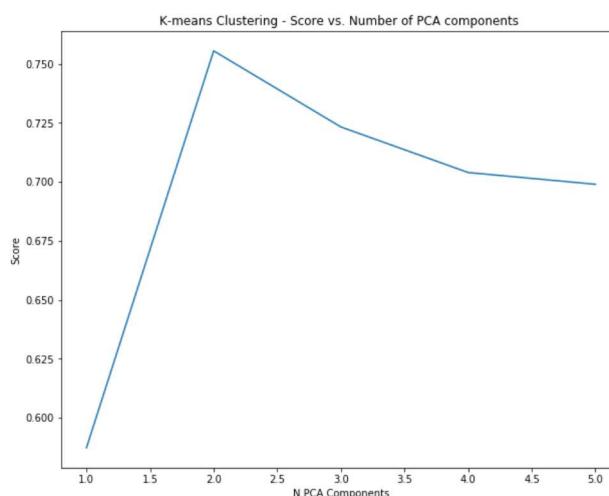
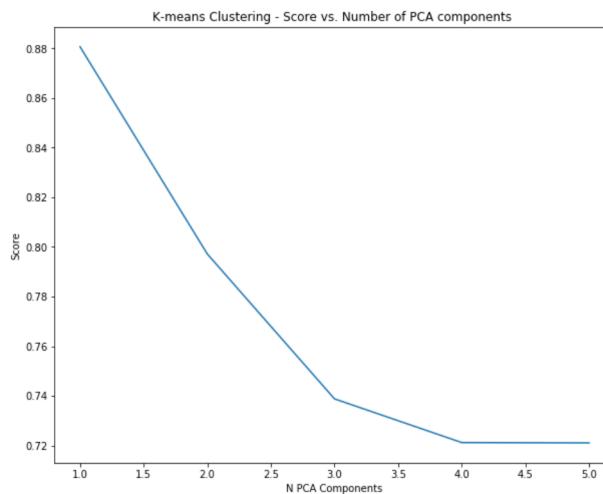
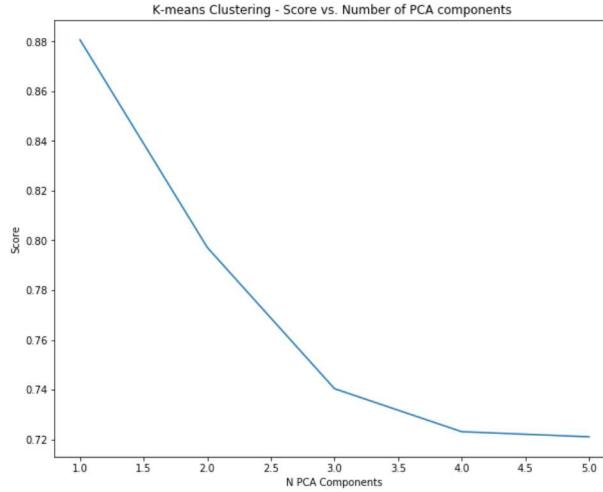


9B. Number of features / PCA Components

```
In [430]: def optimize_kmeans_pcacomponents(df_imgstats, feature_names, n_clusters):  
    """  
    Run the full pipeline (except for visualization) for different number of PCA components  
    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)  
  
    fig = plt.figure(figsize=(10,8))  
    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 [431]: def optimize_hierarchical_pcacomponents(df_imgstats, feature_names, n_clusters):  
    """  
    Run the full pipeline (except for visualization) for different number of PCA components  
    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)  
  
    fig = plt.figure(figsize=(10,8))  
    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()
```

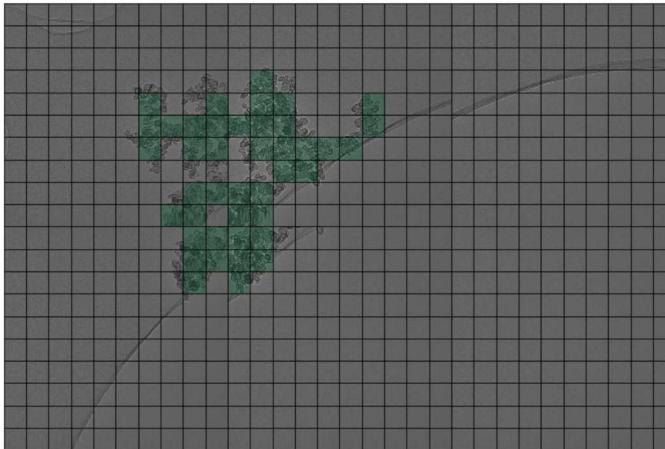
```
In [436]: optimize_kmeans_pcacomponents(df, feature_names, n_clusters=2)
optimize_kmeans_pcacomponents(df, feature_names, n_clusters=2)
optimize_kmeans_pcacomponents(df, feature_names, n_clusters=4)
```



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

```
In [440]: #score = run_kmeans_pipeline(df, feature_names, 2, 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), no_borders=False, heatdependent_opacity=True)
```

Heats from: kmeans

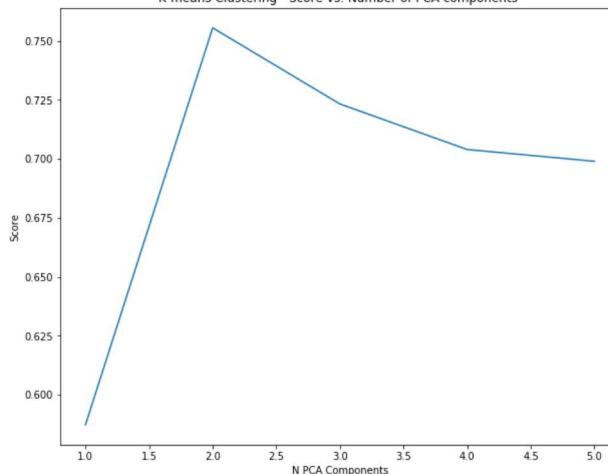


Indeed not bad. Maybe not so surprise as in early EDA we saw that on this particular set the standard deviation is very informative (we used it for manual labelling)

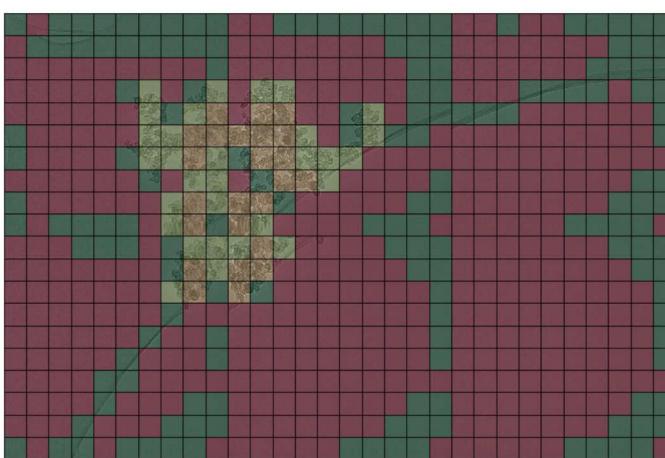
A 'manual grid search' showed for 4 clusters the same

```
In [434]: 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))
```

K-means Clustering - Score vs. Number of PCA components



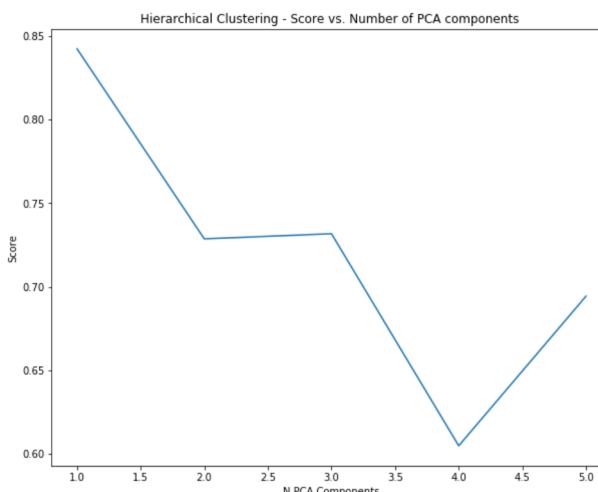
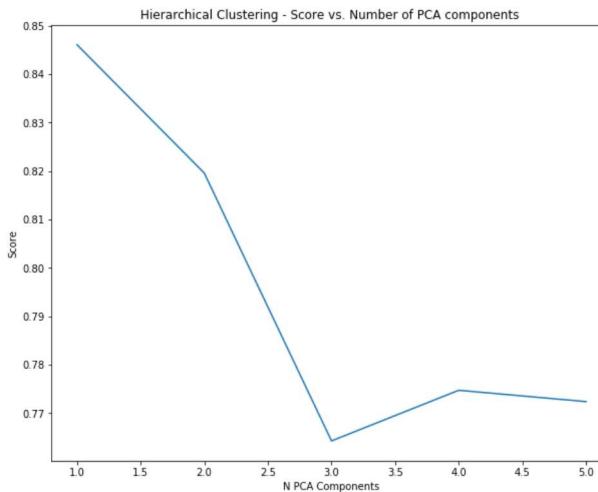
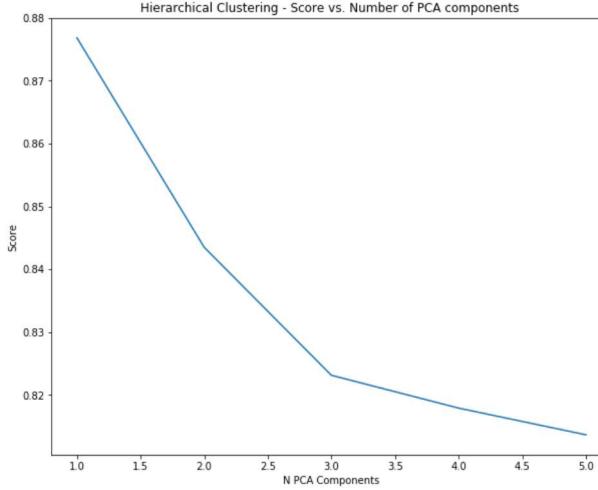
Heats from: kmeans



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

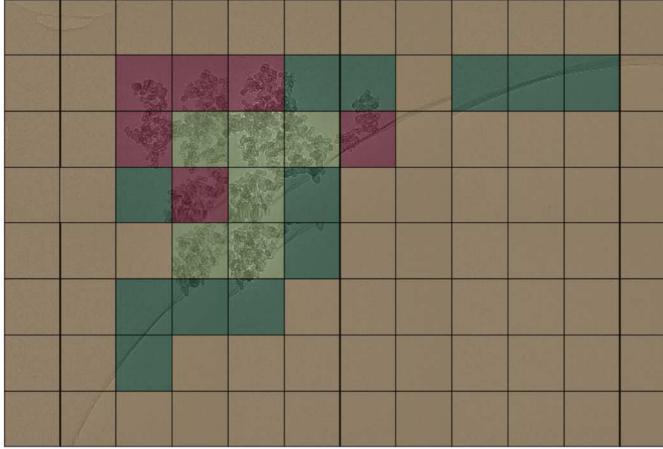


Don't know what these graphs really say; as the score is assessing cluster cohesion based on the last transformation in the pre-processing, it can be expected that less components are more separated. Maybe need to look at the 'elbows' here?

```
In [41]: # visualize for 4 clusters with 1 pca component:
```

```
In [42]: 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))

Heats from: hierarchical
```



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 [392]: fig = plt.figure(figsize=(12,10))

In [393]: 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} %\r".format(progress))

        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_hierarchical))

        counter += 1
    sys.stdout.write("Done.\n")
    #sys.stdout.flush()

    plotdata = list(zip(*results))

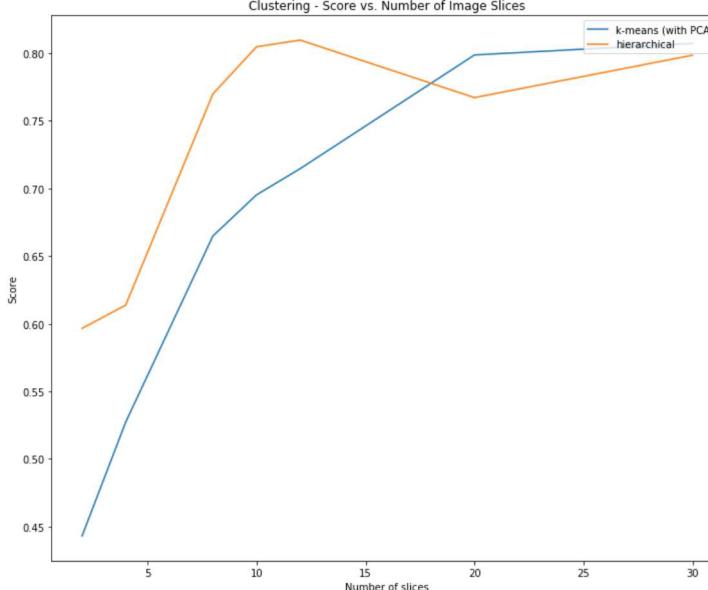
    fig = plt.figure(figsize=(12,10))
    plt.plot(plotdata[1], plotdata[2], label="k-means (with PCA)")
    plt.plot(plotdata[1], plotdata[3], label="hierarchical")
    plt.title('Clustering - Score vs. Number of Image Slices')
    plt.xlabel('Number of slices')
    plt.ylabel('Score')
    plt.legend(loc='upper right')
    plt.show()

    return results
```

```
In [394]: slice_opt_results_3clusters = optimize_slices(datafolder, feature_funcs, n_clusters=3, list_n_slices=[2,4,8,10,12,20,30])
```

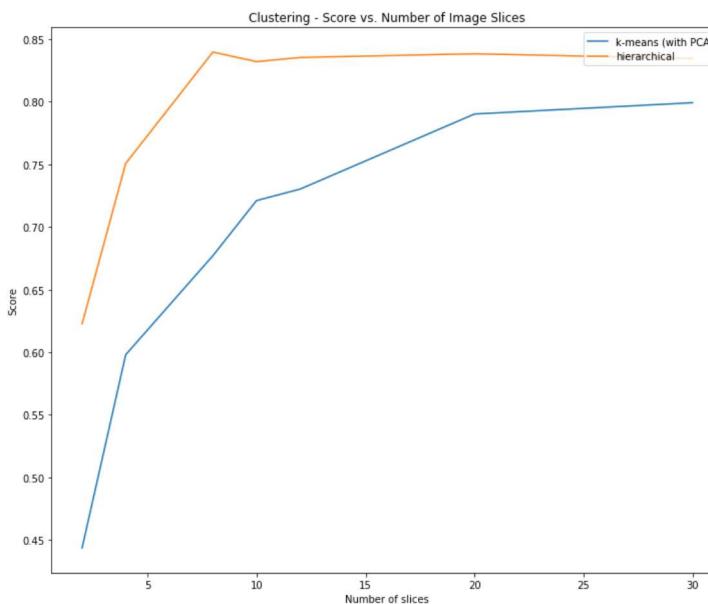
Running slice optimization for 3 clusters:
Done.%

<Figure size 864x720 with 0 Axes>



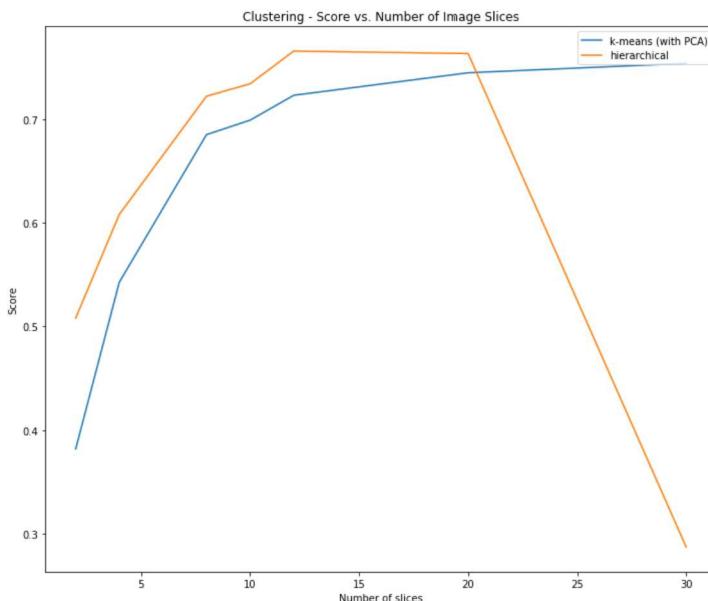
```
In [395]: slice_opt_results_2clusters = optimize_slices(datafolder, feature_funcs, n_clusters=2, list_n_slices=[2,4,8,10,12,20,30])
```

Running slice optimization for 2 clusters:
Done.%



```
In [396]: slice_opt_results_4clusters = optimize_slices(datafolder, feature_funcs, n_clusters=4, list_n_slices=[2,4,8,10,12,20,30])
```

Running slice optimization for 4 clusters:
Done.%



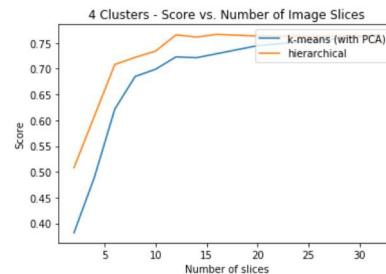
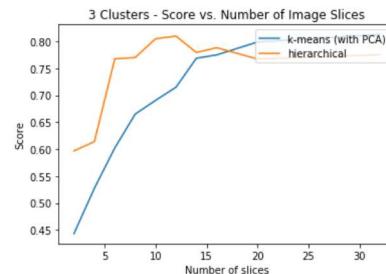
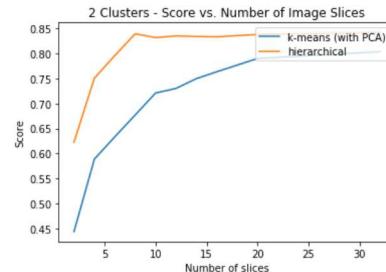
Interesting, optimum number of slices is around 10 for 3 clusters, and even higher number of patches for 4 clusters.

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

```
In [46]: 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} %\r".format(progress))
        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_hierarchical))
        counter += 1
    sys.stdout.write("Done.\r")
    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 [47]: slice_grid_search = gridsearch_slices(datafolder, feature_funcs, list_n_clusters=[2,3,4], list_n_slices=[2,4,6,8,10,12,14,16,20,32])
```

```
Running slice-#clusters grid-search...
Done.%
```



(I once 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 [48]: 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)
```

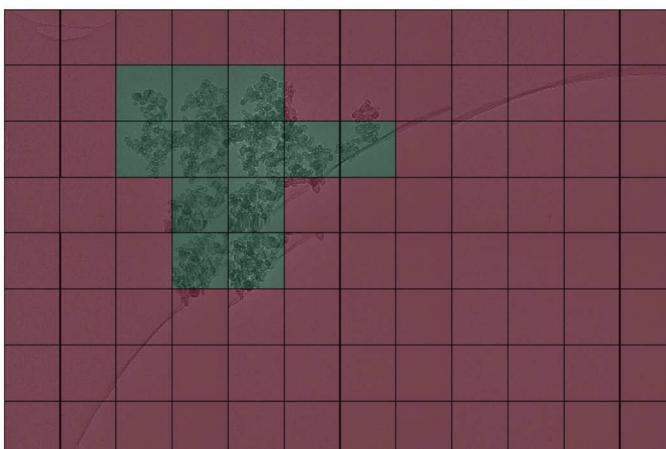
```
In [49]: def show_multiple_from_slicegridsearch(results, list_num_clust, list_num_slices, n_img_rows, n_img_cols,
                                             heat_col='kmeans', fig_size=(12,10)):
    for n_slices in list_num_slices:
        for n_clust in list_num_clust:
            show_from_slicegridsearch(results, n_clust, n_slices, n_img_rows, n_img_cols, heat_col, fig_size=fig_size)
```

```
In [50]: 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))
```

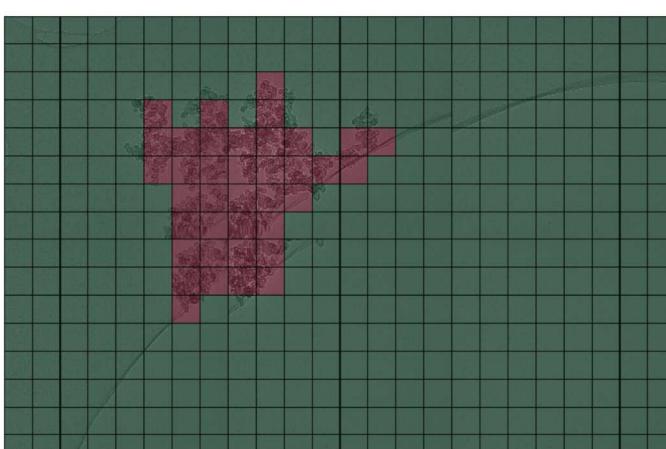
Heats from: kmeans - 2x2 slices, 2 clusters



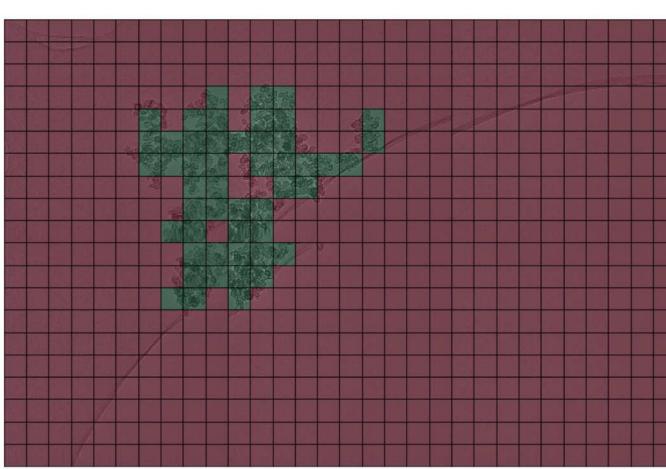
Heats from: kmeans - 4x4 slices, 2 clusters



Heats from: kmeans - 8x8 slices, 2 clusters



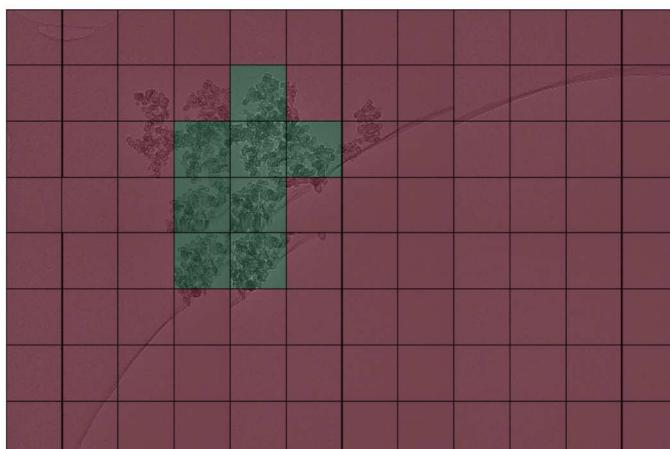
Heats from: kmeans - 10x10 slices, 2 clusters



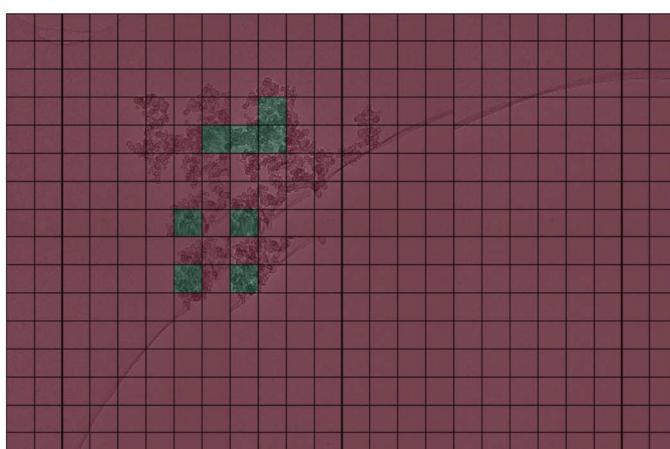
```
In [51]: show_multiple_from_slicegridsearch(slice_grid_search, [2], [2,4,8, 10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical', fig_size=(10,8))  
Heats from: hierarchical - 2x2 slices, 2 clusters
```



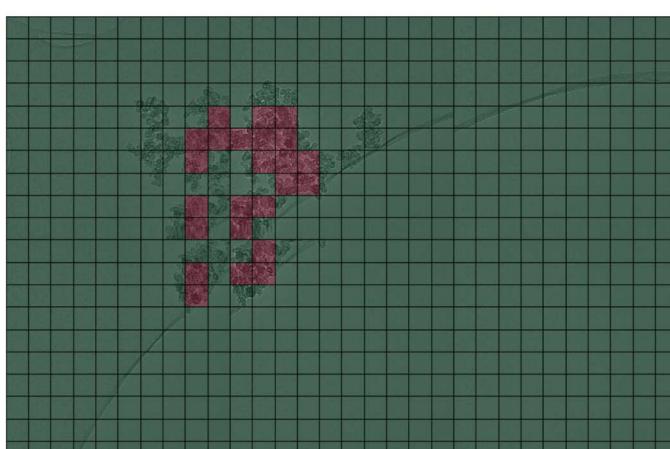
Heats from: hierarchical - 2x2 slices, 2 clusters



Heats from: hierarchical - 4x4 slices, 2 clusters



Heats from: hierarchical - 8x8 slices, 2 clusters



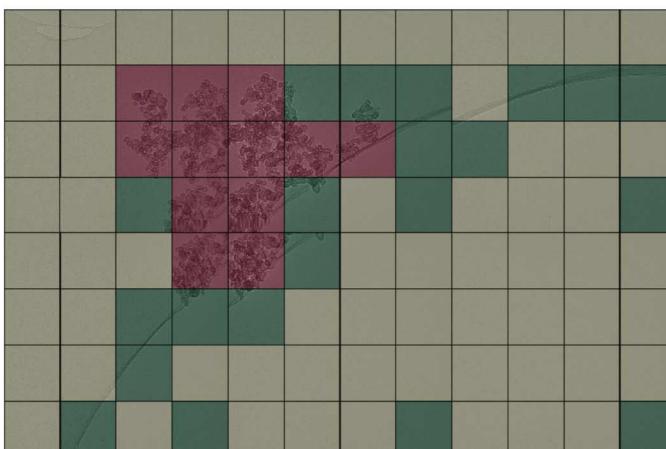
10c. Look at some heatmaps for 3 clusters

```
In [52]: 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))
```

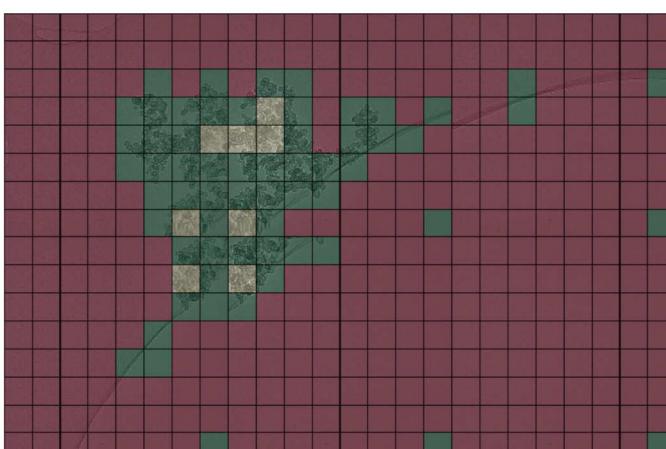
Heats from: kmeans - 2x2 slices, 3 clusters



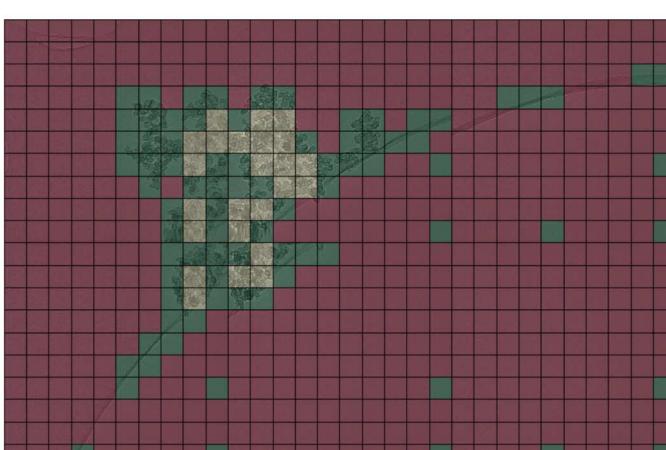
Heats from: kmeans - 4x4 slices, 3 clusters



Heats from: kmeans - 8x8 slices, 3 clusters



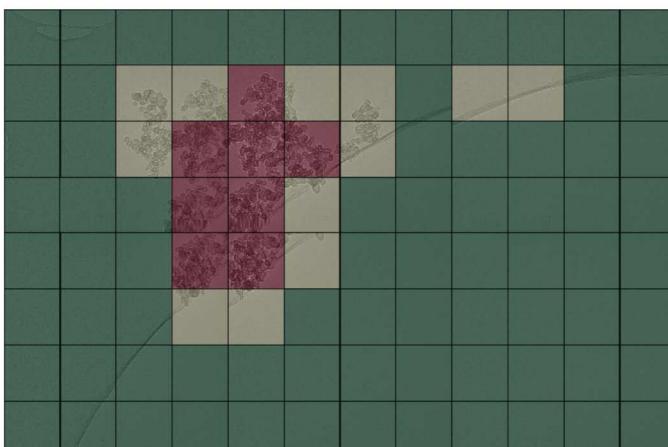
Heats from: kmeans - 10x10 slices, 3 clusters



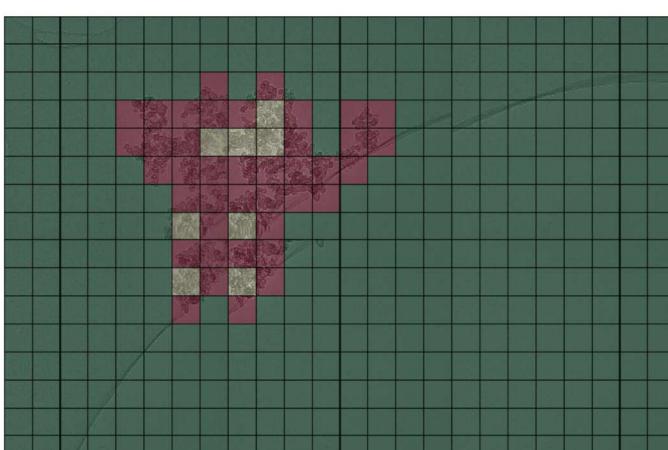
```
In [53]: show_multiple_from_slicegridsearch(slice_grid_search, [3], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical', fig_size=(10,8))  
Heats from: hierarchical - 2x2 slices, 3 clusters
```



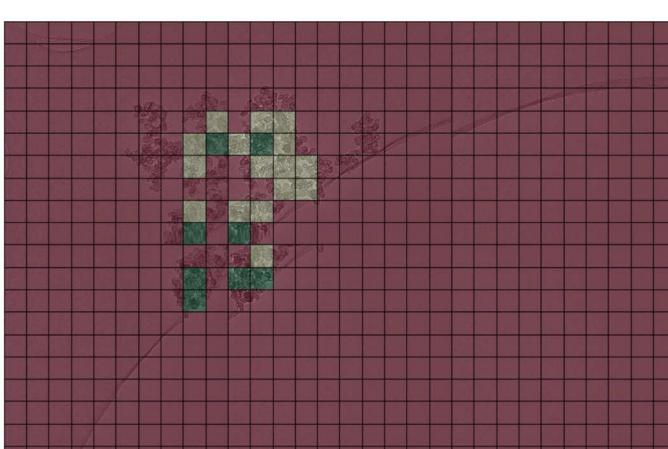
Heats from: hierarchical - 2x2 slices, 3 clusters



Heats from: hierarchical - 4x4 slices, 3 clusters



Heats from: hierarchical - 8x8 slices, 3 clusters



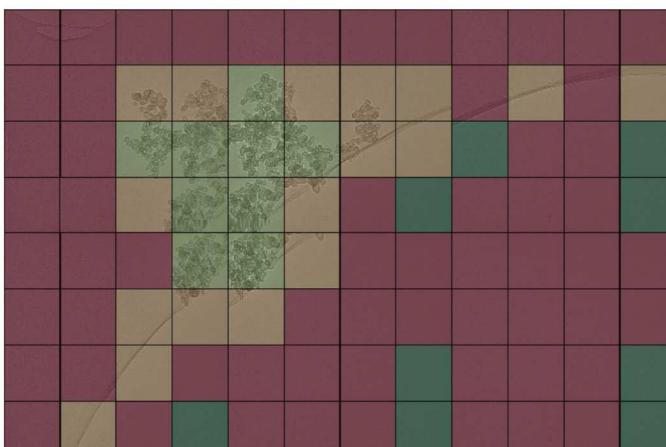
10D. Look at some heatmaps for 4 clusters

```
In [54]: 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))
```

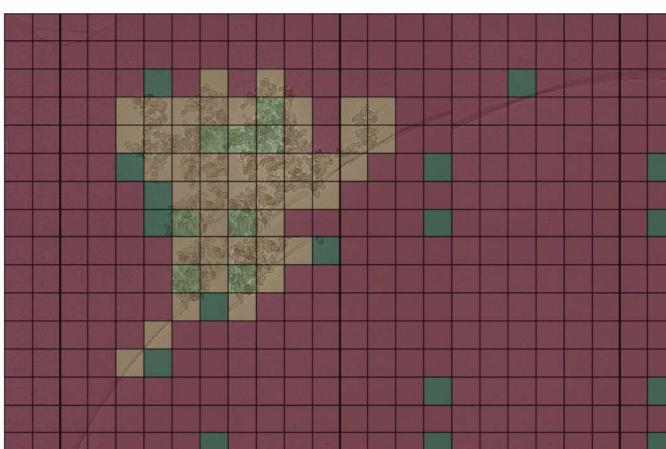
Heats from: kmeans - 2x2 slices, 4 clusters



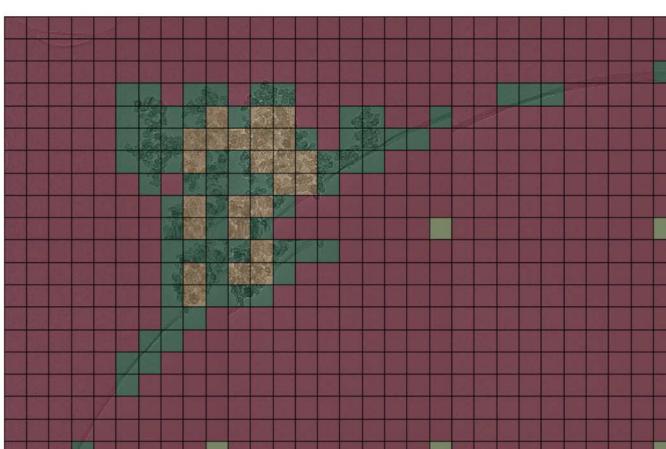
Heats from: kmeans - 4x4 slices, 4 clusters



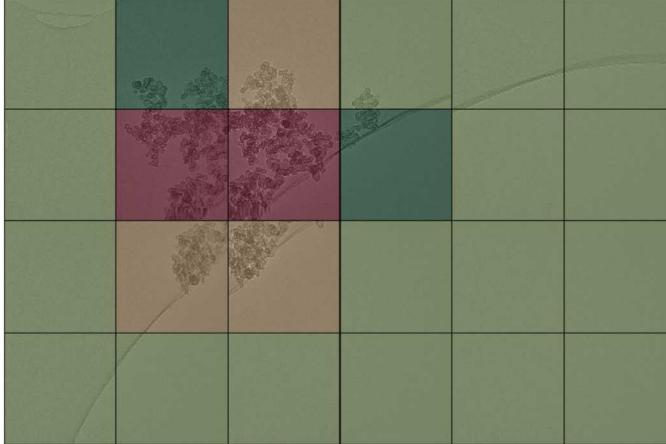
Heats from: kmeans - 8x8 slices, 4 clusters



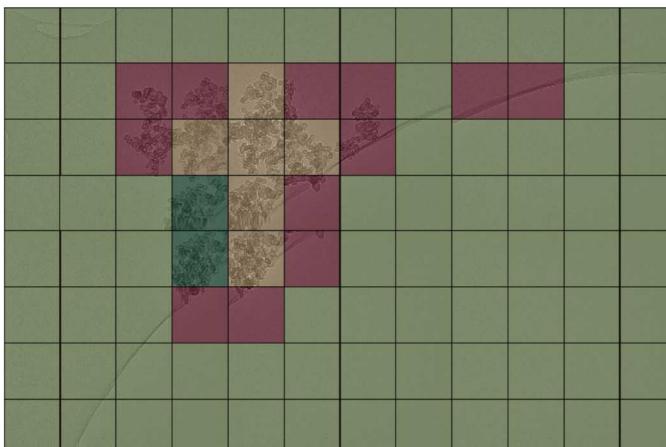
Heats from: kmeans - 10x10 slices, 4 clusters



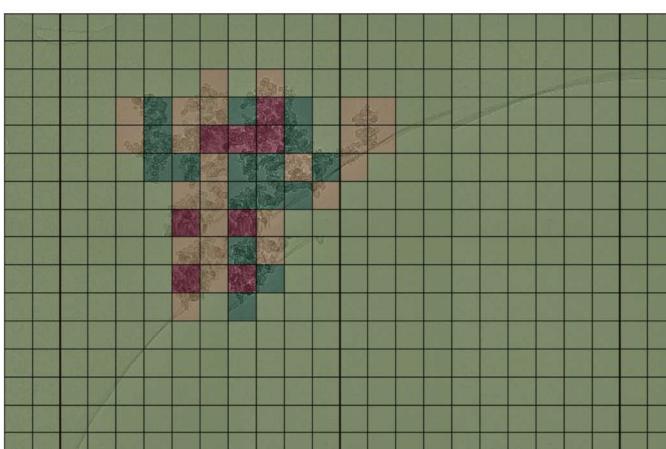
```
In [55]: show_multiple_from_slicegridsearch(slice_grid_search, [4], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col='hierarchical', fig_size=(10,8))  
Heats from: hierarchical - 2x2 slices, 4 clusters
```



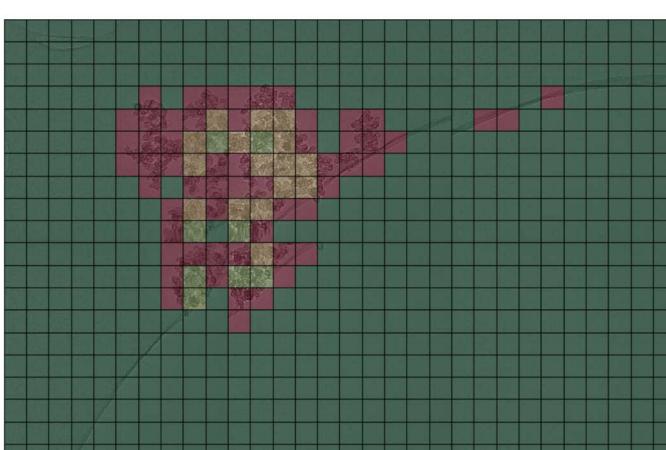
Heats from: hierarchical - 2x2 slices, 4 clusters



Heats from: hierarchical - 4x4 slices, 4 clusters



Heats from: hierarchical - 8x8 slices, 4 clusters



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 !)

Extra Visualizations for report (Nov 2018)

```
In [218]: 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)
```

Gradually zooming in; for 2, 3 and 4 clusters

```
In [558]: all_imgfiles = imgutils.scanimgdir(datafolder, '.tif')[['filename']].tolist()
img2x2 = [all_imgfiles[0], all_imgfiles[1],all_imgfiles[3],all_imgfiles[4]]
img1 = all_imgfiles[0]
img2 = all_imgfiles[1]
```

Zoomed out

```
In [603]: df2_out = imgutils.slicestats(all_imgfiles, 4, 4, feature_funcs)
run_ml_PIPELINES(df2_out, feature_names, n_clusters = 2, standardize=True, use_pca=True)

df3_out = imgutils.slicestats(all_imgfiles, 4, 4, feature_funcs)
run_ml_PIPELINES(df3_out, feature_names, n_clusters = 3, standardize=True, use_pca=True)

n_clusters = 4
df4_out = imgutils.slicestats(all_imgfiles, 4, 4, feature_funcs)
run_ml_PIPELINES(df4_out, feature_names, n_clusters = 4, standardize=True, use_pca=True)

Executing clustering pipelines...
Done

Clustering Scores:
K-means:  0.5886428860898483
Hierarchical:  0.5886428860898483
Executing clustering pipelines...
Done

Clustering Scores:
K-means:  0.5251916981567152
Hierarchical:  0.5543189694833234
Executing clustering pipelines...
Done

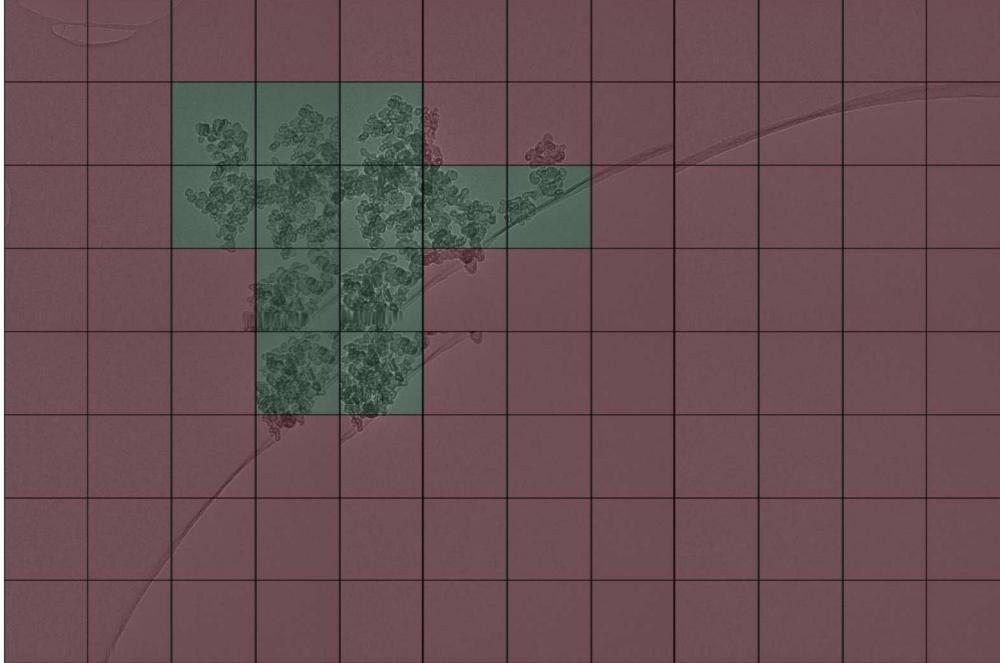
Clustering Scores:
K-means:  0.47896152275481274
Hierarchical:  0.5252113084761886
```

```
In [614]: #swap_clusters(df2_out, 'kmeans', 1,2)
```

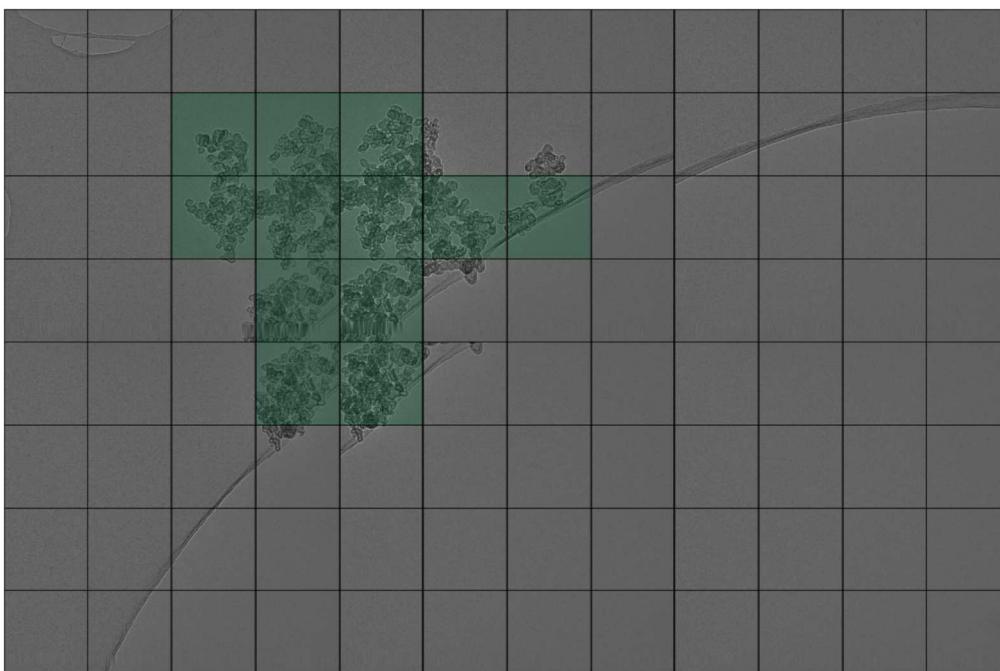
2 clusters

```
In [607]: imgutils.show_large_heatmap(df2_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df2_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

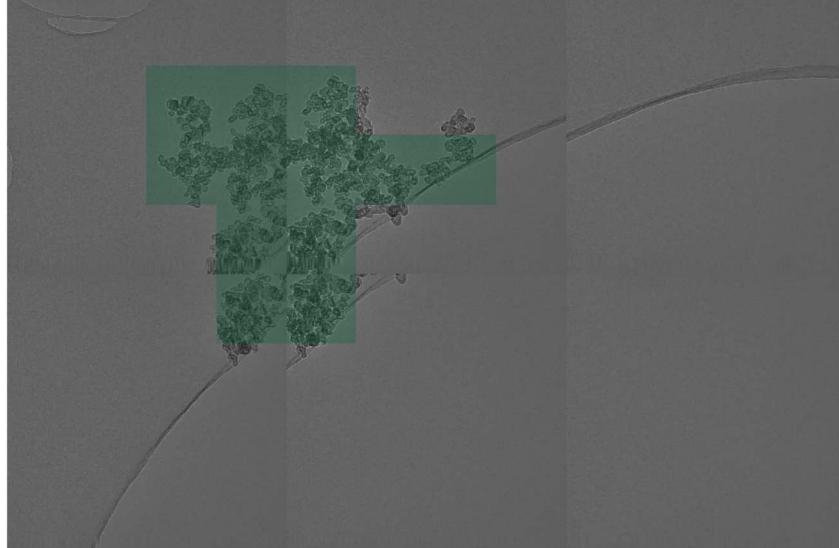
Heats from: kmeans



Heats from: kmeans



```
In [721]: imgutils.show_large_heatmap(df2_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=(12,10), heatdependent_opacity=True, no_borders=True)  
Heats from: kmeans
```

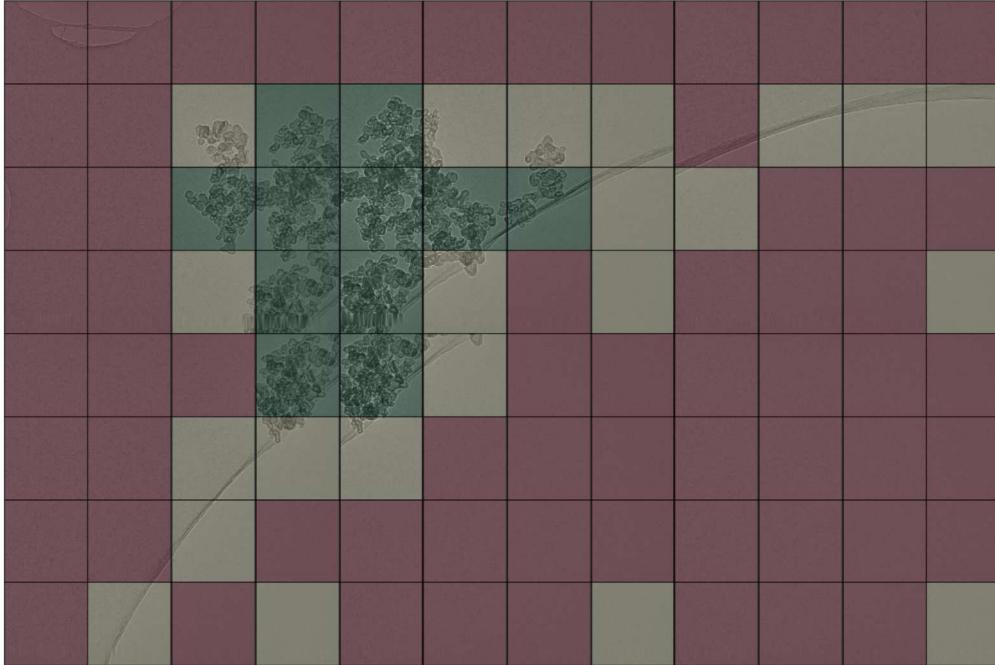


```
In [612]: #swap_clusters(df3_out, 'kmeans', 1, 2)
```

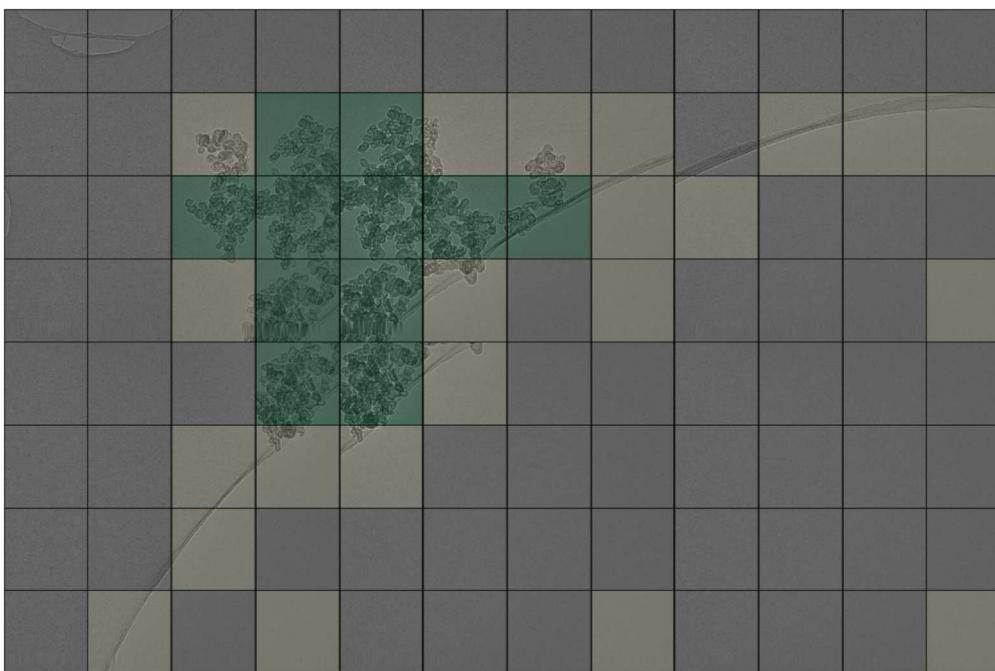
3 clusters

```
In [613]: imgutils.show_large_heatmap(df3_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df3_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans

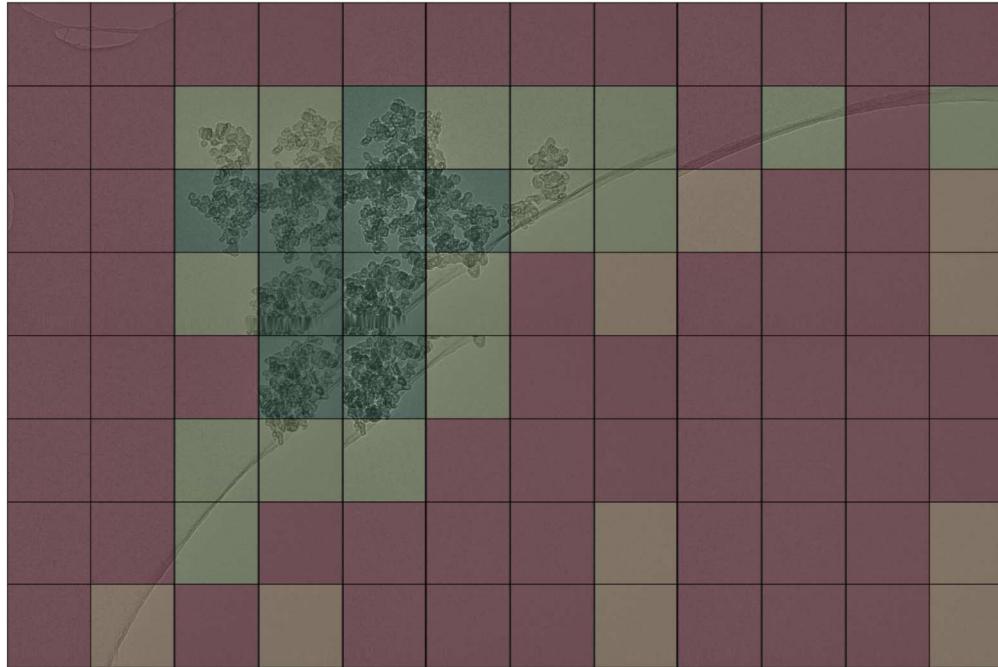


4 clusters

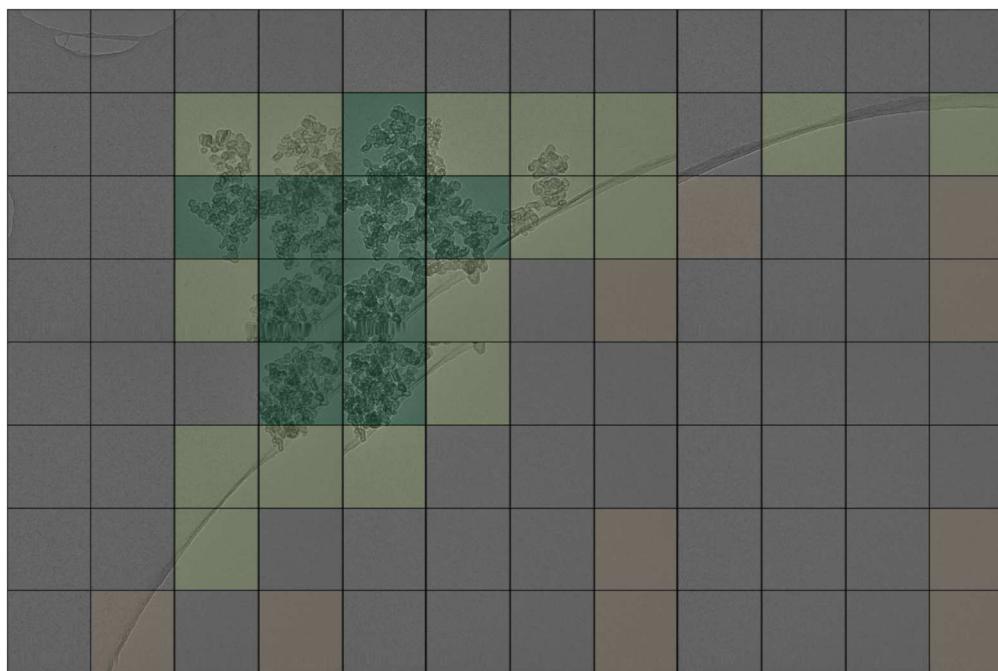
```
In [643]: #swap_clusters(df4_out, 'kmeans', 1, 2)
```

```
In [644]: imgutils.show_large_heatmap(df4_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df4_out, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans



Zoomed Intermediate

```
In [685]: df2_med = imgutils.slicestats(all_imgfiles, 8, 8, feature_funcs)
run_ml_pipelines(df2_med, feature_names, n_clusters = 3, standardize=True, use_pca=True)

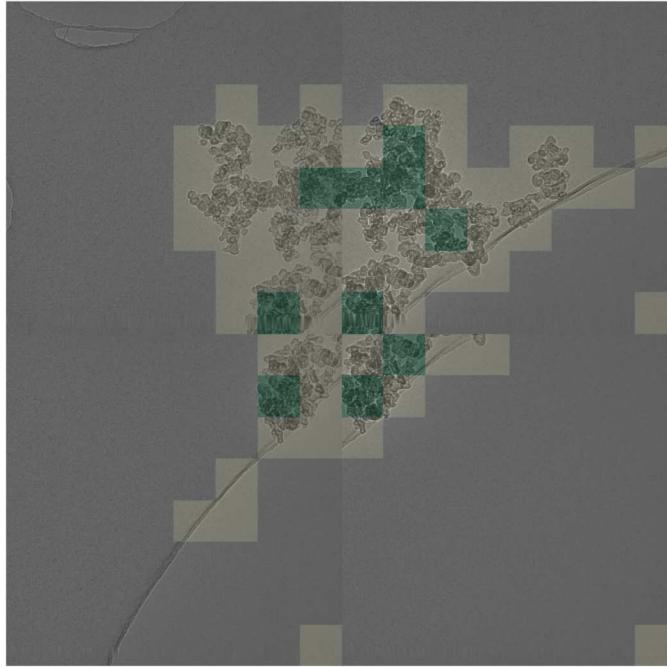
Executing clustering pipelines...
Done
```

Clustering Scores:
K-means: 0.6623808566364214
Hierarchical: 0.5944302326769709

```
In [689]: swap_clusters(df2_med, 'kmeans', 0,1)
```

```
In [690]: imgutils.show_large_heatmap(df2_med, 'kmeans', img2x2, n_rows=2, n_cols=2, fig_size=s, heatdependent_opacity=True, no_borders=True)
```

Heats from: kmeans



Zoomed in / refined

```
In [618]: df2_in = imgutils.slicestats(all_imgfiles, 20, 20, feature_funcs)
run_ml_pipelines(df2_in, feature_names, n_clusters = 2, standardize=True, use_pca=True)

df3_in = imgutils.slicestats(all_imgfiles, 20, 20, feature_funcs)
run_ml_pipelines(df3_in, feature_names, n_clusters = 3, standardize=True, use_pca=True)

df4_in = imgutils.slicestats(all_imgfiles, 20, 20, feature_funcs)
run_ml_pipelines(df4_in, feature_names, n_clusters = 4, standardize=True, use_pca=True)
```

Executing clustering pipelines...
Done

Clustering Scores:
K-means: 0.7901963892977328
Hierarchical: 0.92278381185474
Executing clustering pipelines...
Done

Clustering Scores:
K-means: 0.7987701441108157
Hierarchical: 0.7930404098862842
Executing clustering pipelines...
Done

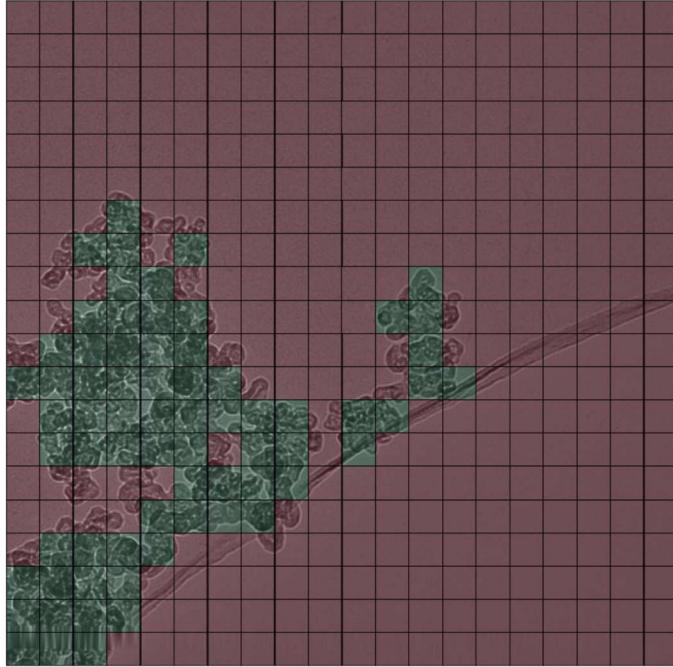
Clustering Scores:
K-means: 0.7446185778518968
Hierarchical: 0.801532587150146

2 clusters

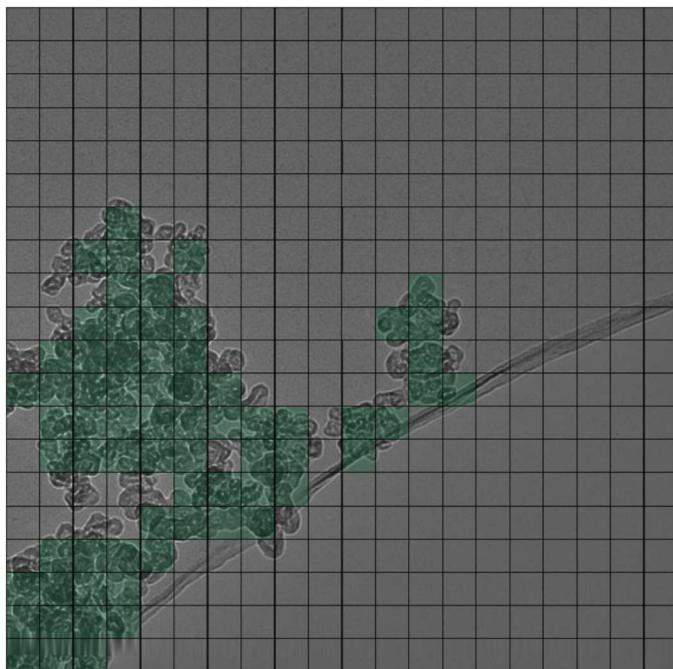
```
In [619]: #swap_clusters(df2_in, 'kmeans', 1,2)
```

```
In [668]: imgutils.show_large_heatmap(df2_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df2_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans

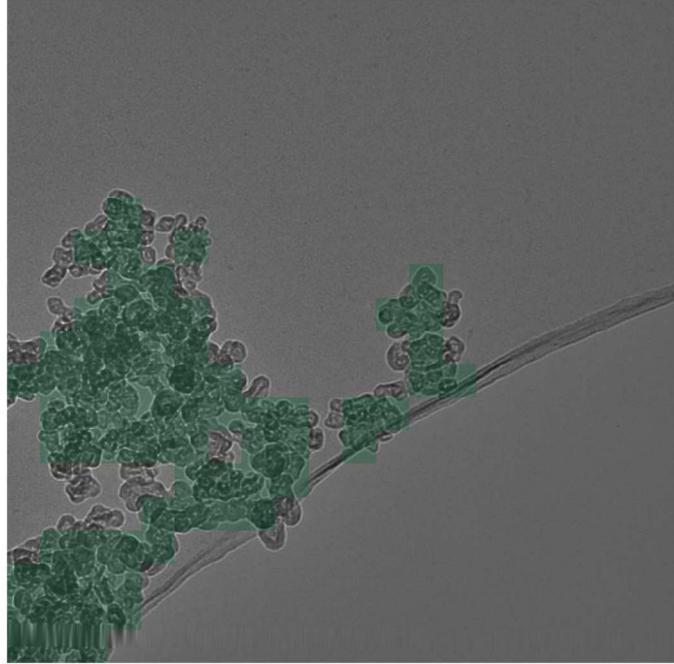


Heats from: kmeans



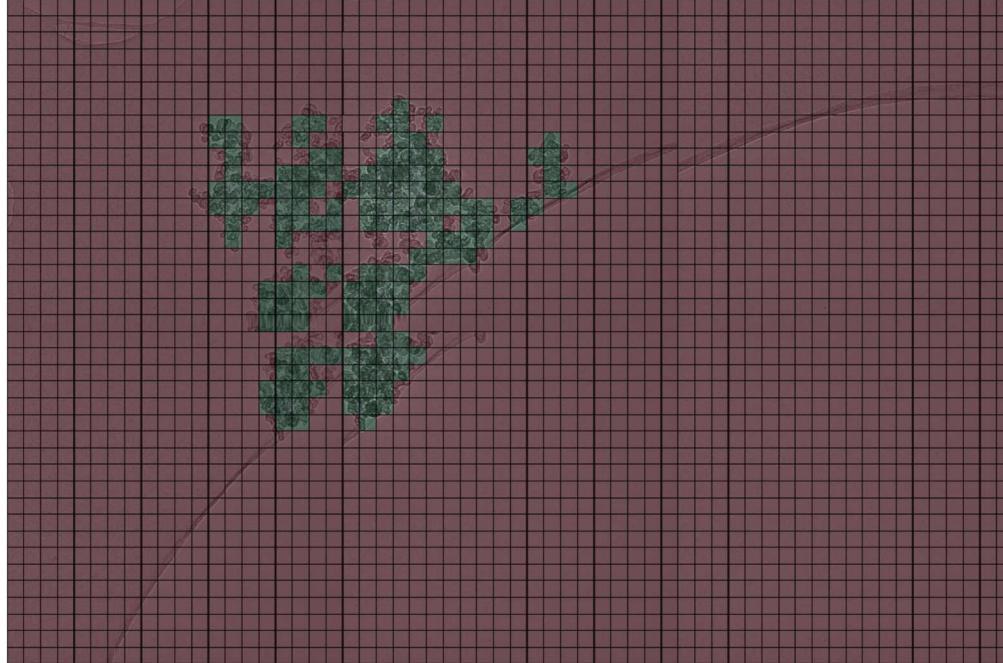
```
In [670]: imgutils.show_large_heatmap(df2_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True, no_borders=True)
```

Heats from: kmeans

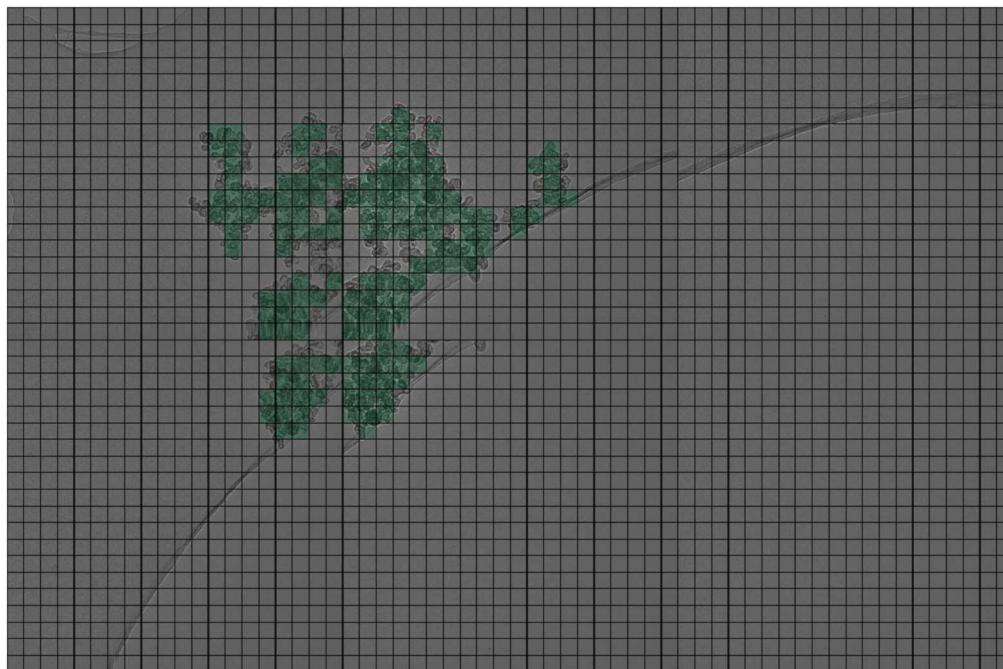


```
In [620]: imgutils.show_large_heatmap(df2_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df2_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans

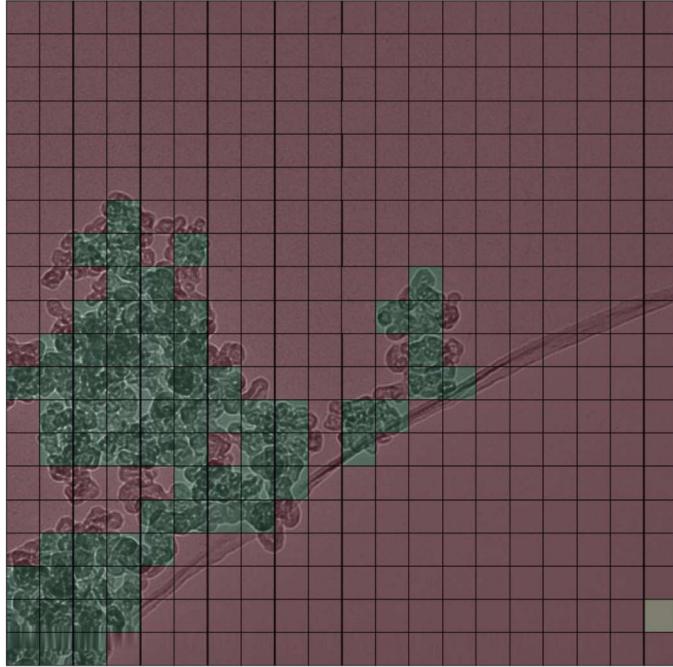


3 clusters

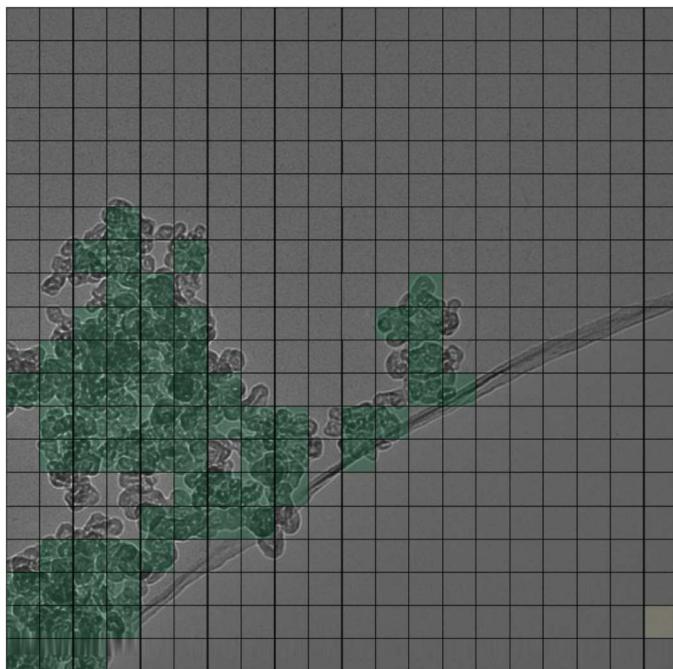
```
In [648]: #swap_clusters(df3_in, 'kmeans', 1,2)
```

```
In [667]: imgutils.show_large_heatmap(df3_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df3_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans

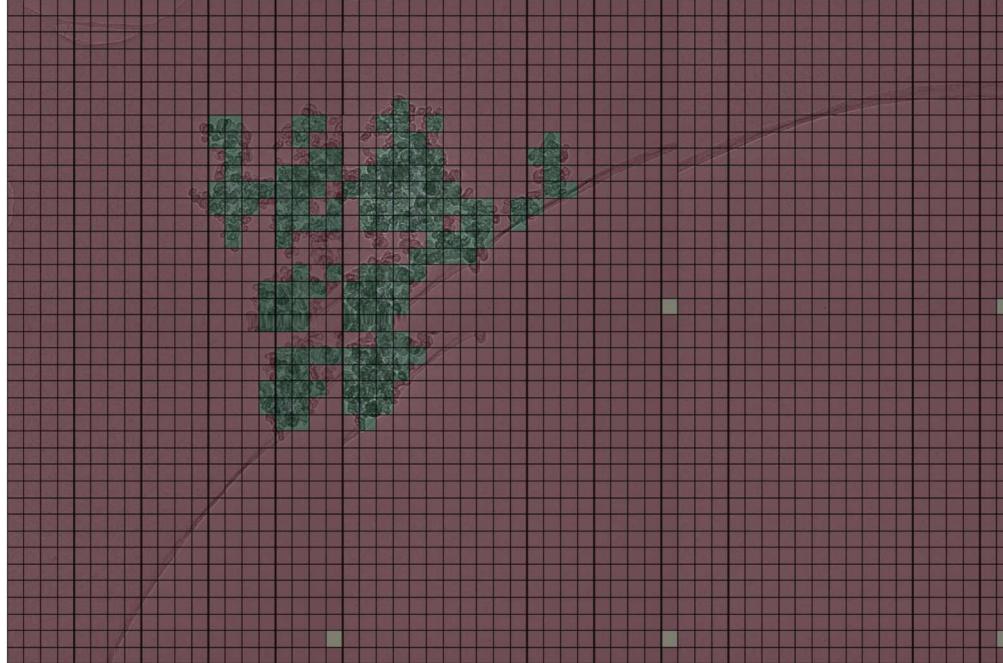


Heats from: kmeans

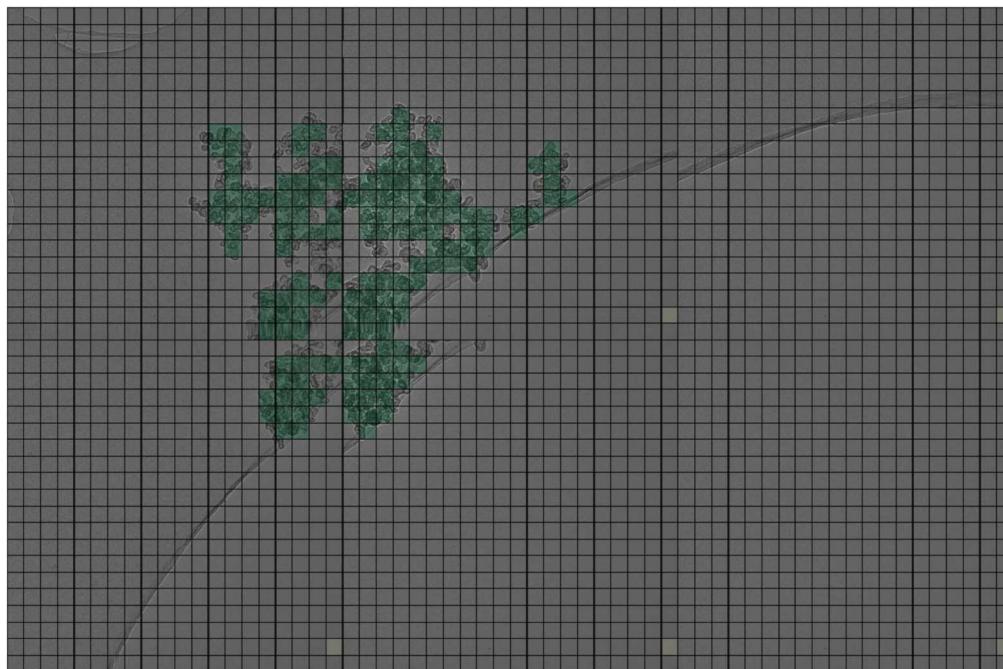


```
In [669]: imgutils.show_large_heatmap(df3_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df3_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans

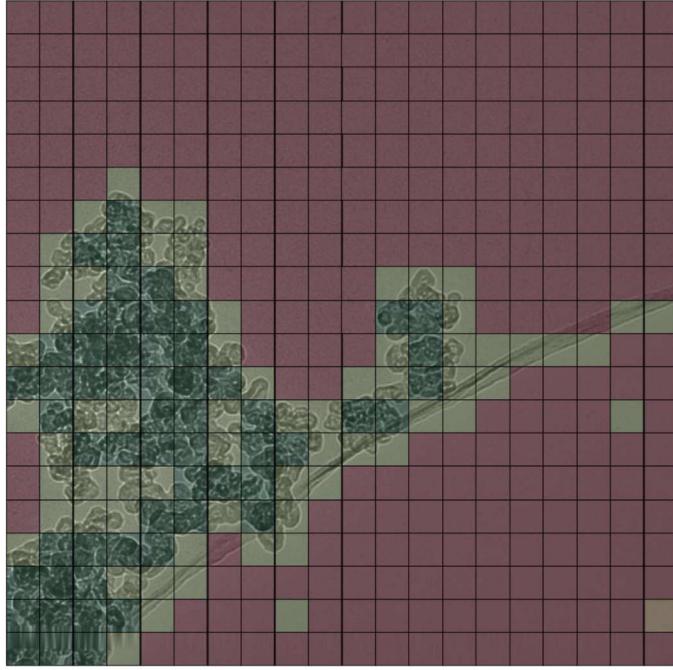


4 clusters

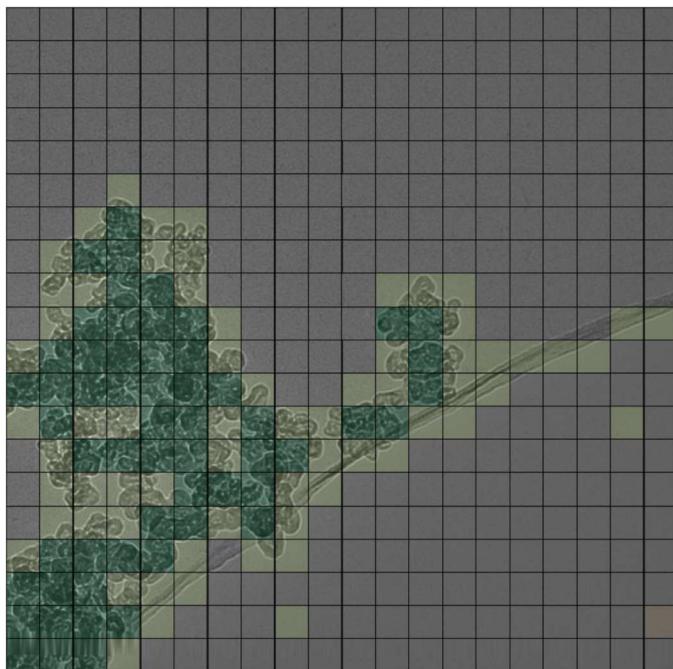
```
In [664]: #change_clusternums(df4_in, 'kmeans', { 3 : 2, 4:1, 2:3})
```

```
In [665]: imgutils.show_large_heatmap(df4_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df4_in, 'kmeans', [img2], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans

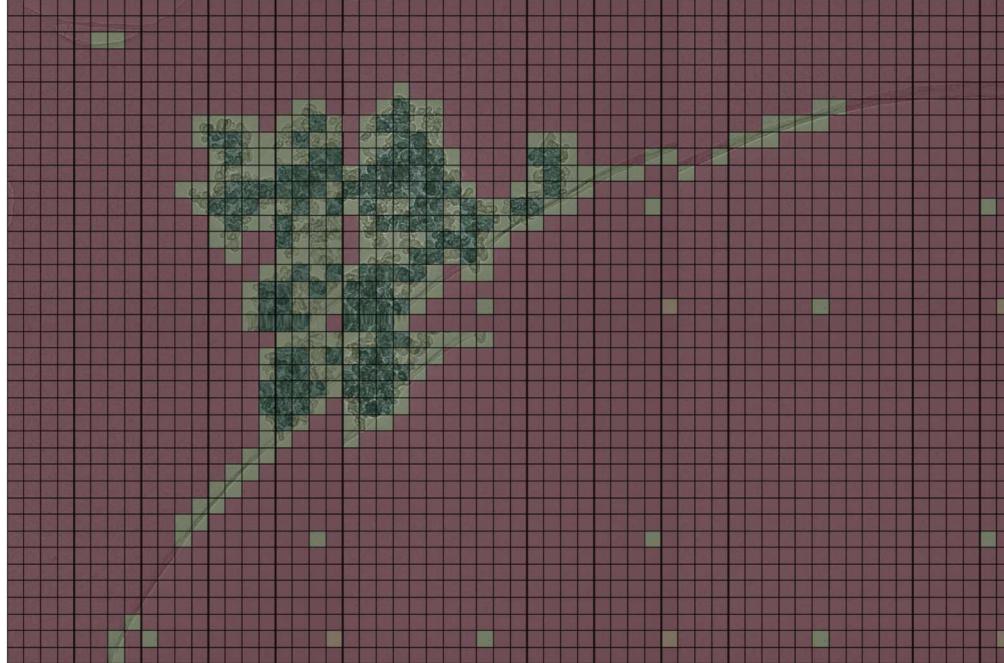


Heats from: kmeans

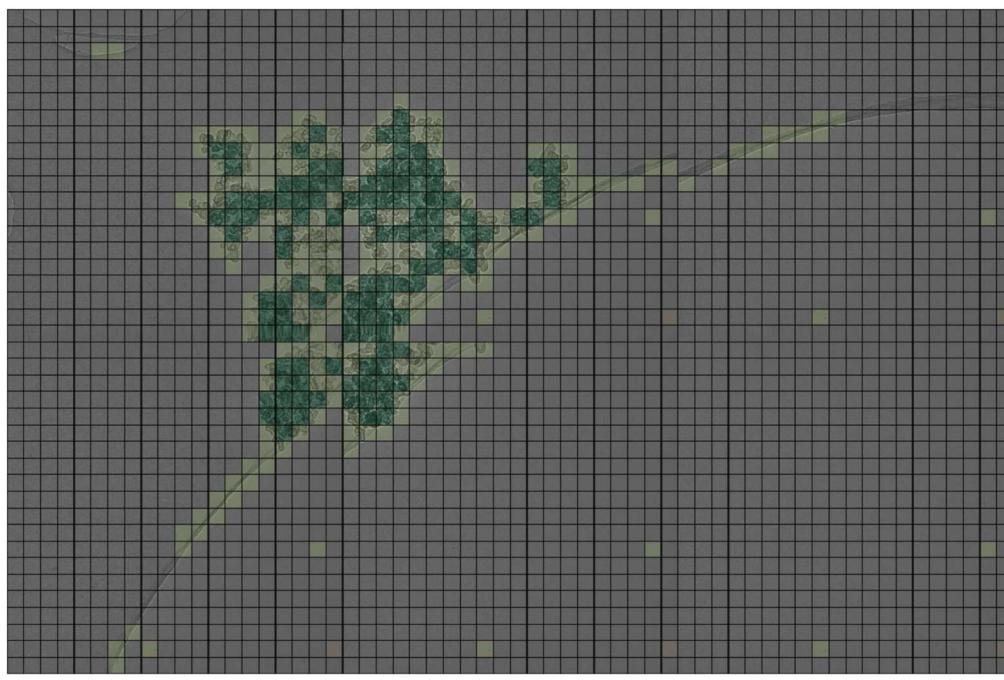


```
In [666]: imgutils.show_large_heatmap(df4_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df4_in, 'kmeans', all_imgfiles, n_rows=2, n_cols=3, fig_size=s, heatdependent_opacity=True)
```

Heats from: kmeans



Heats from: kmeans



Other similar tile set

```
In [727]: imgfiles8 = imgutils.scanimgdir('../data/Crystals_Apr_12/Tileset8', '.tif')['filename'].tolist()
print(imgfiles8)
s2 = (12,10)
['..\\data\\Crystals_Apr_12\\Tileset8\\Tile_001-001-000_0-000.tif', '..\\data\\Crystals_Apr_12\\Tileset8\\Tile_001-002-000_0-000.tif', '..\\data\\Crystals_Apr_12\\Tileset8\\Tile_001-003-000_0-000.tif']
```

```
In [758]: df8 = imgutils.slicestats(imgfiles8, 6, 6, feature_funcs)
run_ml_pipelines(df8, feature_names, n_clusters = 2, standardize=True, use_pca=True)

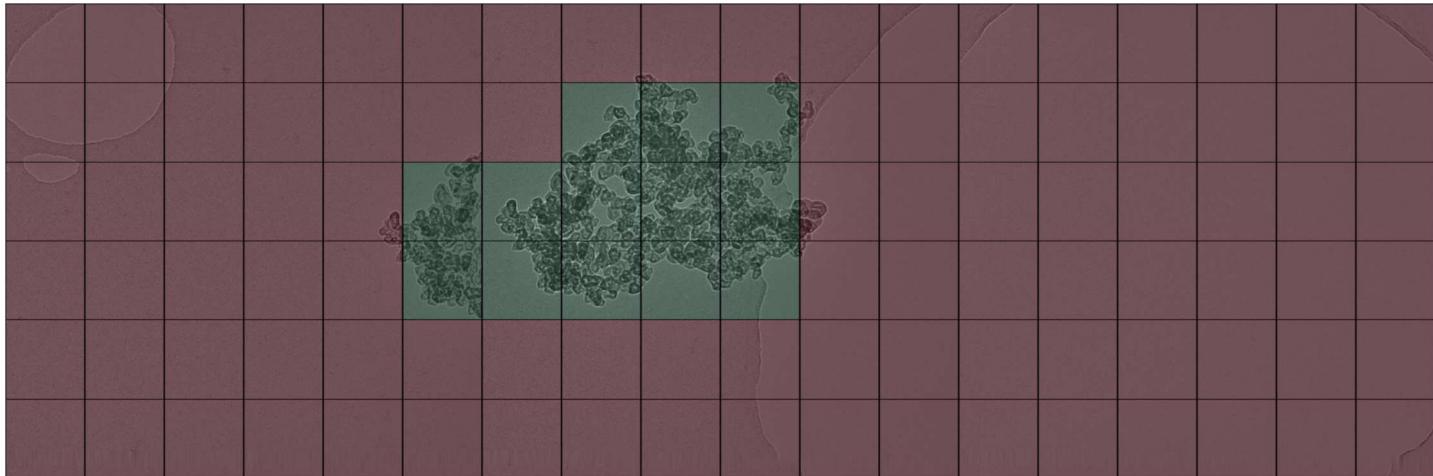
Executing clustering pipelines...
Done
```

Clustering Scores:
K-means: 0.6289743035702563
Hierarchical: 0.7282552485033036

```
In [754]: swap_clusters(df8, 'kmeans', 0,1)
```

```
In [759]: imgutils.show_large_heatmap(df8, 'kmeans', imgfiles8, n_rows=1, n_cols=3, fig_size=s2, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df8, 'kmeans', imgfiles8, n_rows=1, n_cols=3, fig_size=s2, heatdependent_opacity=True, no_borders=True)
```

Heats from: kmeans

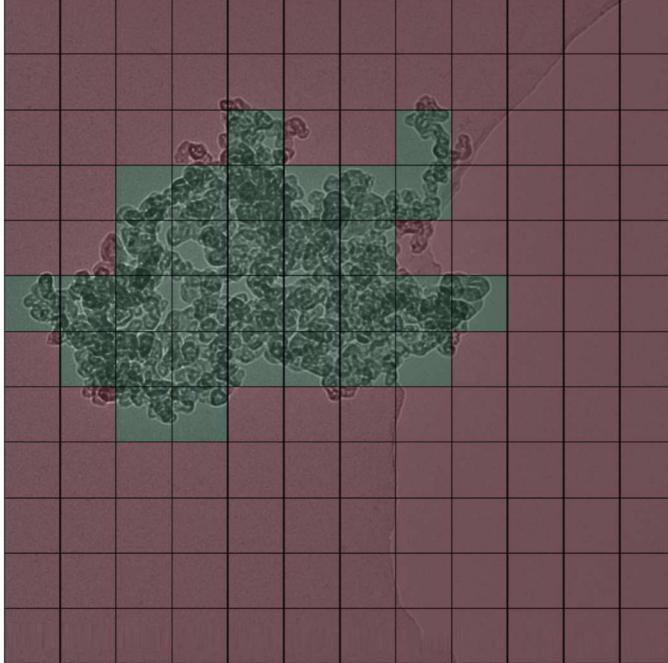


Heats from: kmeans

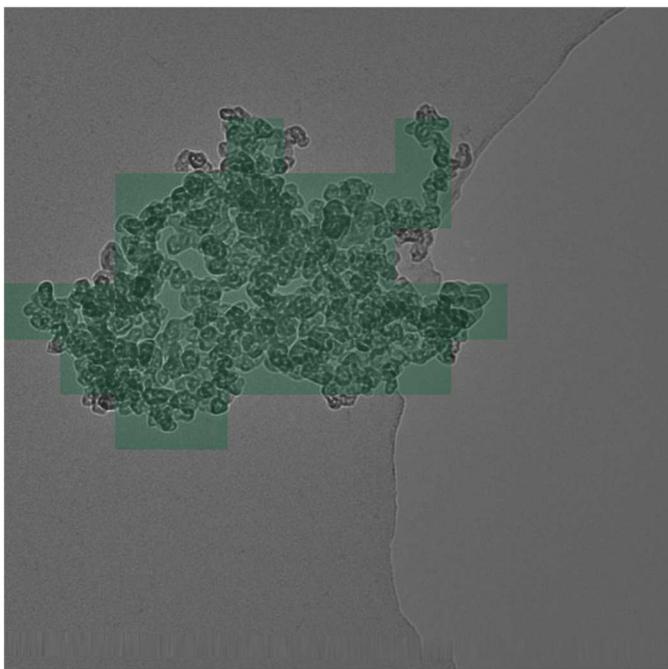


```
In [716]: imgutils.show_large_heatmap(df8, 'kmeans', [imgfiles8[1]], n_rows=1, n_cols=1, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df8, 'kmeans', [imgfiles8[1]], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True, no_borders=True)
```

Heats from: kmeans



Heats from: kmeans



```
In [760]: df8_fine = imgutils.slicestats(imgfiles8, 20, 20, feature_funcs)
run_ml_pipelines(df8_fine, feature_names, n_clusters = 2, standardize=True, use_pca=True)

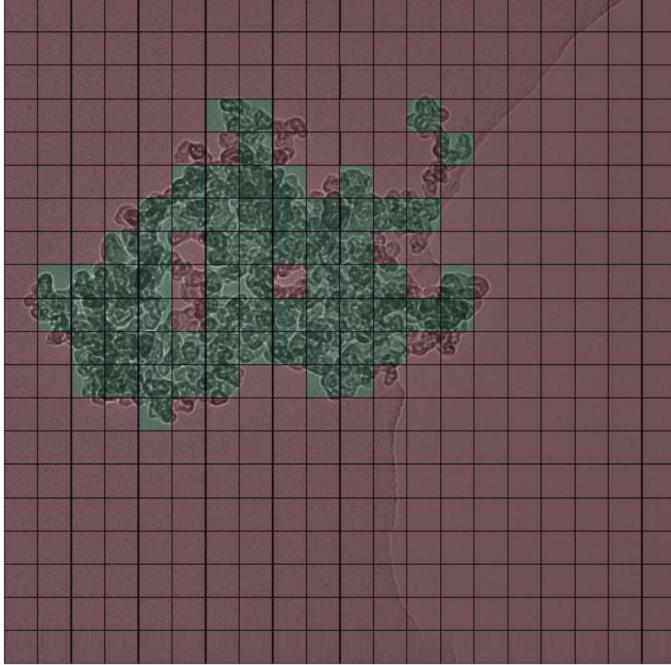
Executing clustering pipelines...
Done
```

Clustering Scores:
K-means: 0.782925201798415
Hierarchical: 0.8820241222775032

```
In [756]: swap_clusters(df8_fine, 'kmeans', 1, 2)
```

```
In [761]: imgutils.show_large_heatmap(df8_fine, 'kmeans', [imgfiles8[1]], n_rows=1, n_cols=1, fig_size=s, opacity=0.2, heatdependent_opacity=False)
imgutils.show_large_heatmap(df8_fine, 'kmeans', [imgfiles8[1]], n_rows=1, n_cols=1, fig_size=s, heatdependent_opacity=True, no_borders=True)
```

Heats from: kmeans



Heats from: kmeans

