Full Pipeline on Polymers (very very hard) - Nov 2018

Created: 04 Nov 2018
Last update: 04 Nov 2018

Goal: Run the full pipeline on the Polymer sample

---

1. Imports

In [382]:
```
# 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 [416]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)
```

Out[416]:
```
<module 'imgutils' from 'C:\\JADS\\SW\\Grad Proj\\sources\\imgutils.py'>
```

In [384]:
```
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)
```

---

2. Data Definitions & Feature Specification

In [391]:
```
# Data:
datafolder = '../data/Polymers_27Sep2018/LowMag_1K_Subset'
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, 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 = 8
default_grid_y = default_grid_x
```

---

3. Import Data & Extract Features

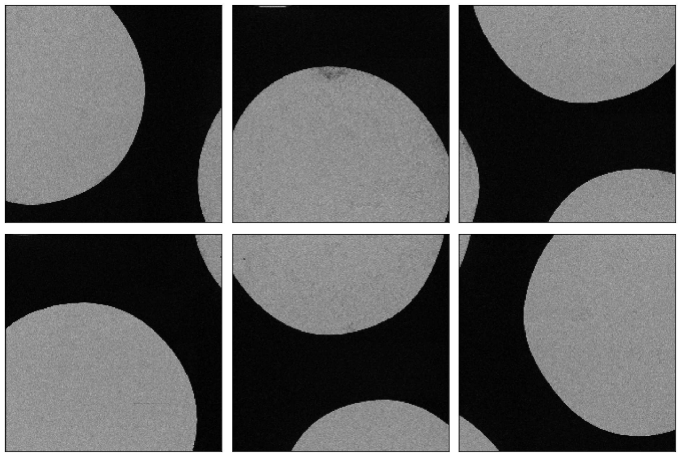
In [387]:
```
# image import:
print("Scanning for images in '{}'...".format(datafolder))
df_imgfiles = imgutils.scanimgdir(datafolder, '.tif')
imgfiles = list(df_imgfiles['filename'])
print("# of images: {} \n".format(len(imgfiles)))

# 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/Polymers_27Sep2018/LowMag_1K_Subset'...
# of images: 6

Feature extraction...
- Slicing up images in 8 x 8 patches.
- Extract statistics from each slice: img_std, img_relstd, img_mean, img_skewness, img_kurtosis, img_mode, img_range
# slices extracted:  384
```

In [6]:
```
# this is a set of 2 x 3 images covering a larger area (a so called 'tile set')
imgutils.showimgset(imgfiles, n_tiles_y, n_tiles_x, fig_size=(12, 8), relspacing=(0.05,0.05))
```



---

4. Machine Learning Pipeline

Hyper parameters

In [7]:
```
# data hyper-parameters
default_n_clusters = 3

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

ML pipeline

In [36]:
```
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 [37]:
```
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)
```

Combine with import

In [38]:
```
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 [39]:
```
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 [40]:
```
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 [41]:
```
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 [42]:
```
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
```

sliding window

In [236]:
```
import sklearn.feature_extraction.image as skimgfeat
import matplotlib.pyplot as plt
import math
import sys

def run_pipeline_windowed(imgfilename, patch_size, n_clusters, downscale_factor=2, return_cluster_image = False,
                          algorithm='kmeans', show_results=True, show_diagnostics=False, show_diagnostics_extra=False):
    """ """

    print("Importing image(s)s...")
    img_full = imgutils.loadtiff(imgfilename)
    img = imgutils.downsample_img(img_full, downscale_factor)
    patches = skimgfeat.extract_patches_2d(img, patch_size)

    sys.stdout.write("Extracting features...")
    patchstats = np.empty((patches.shape[0],4))
    n_progress_update = (int)(patches.shape[0] / 1000)
    for i in range(patches.shape[0]):
        patch = patches[i]
        patchstats[i,0] = np.mean(patch)
        patchstats[i,1] = np.median(patch)
        patchstats[i,2] = np.std(patch)
        patchstats[i,3] = np.max(patch)-np.min(patch)
        if (i % n_progress_update == 0):
            progress = (int)(i*100.0 / patches.shape[0])
            sys.stdout.write("\rExtracting features... {:d} %".format(progress))
            sys.stdout.flush()
    sys.stdout.write("\rExtracting features... 100 %\n")
    sys.stdout.flush()

    print("Clustering into {} clusters...".format(n_clusters))
    kmeans = KMeans(algorithm='auto', n_clusters=n_clusters, n_init=10, init='k-means++')
    hierarch = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete')
    if (algorithm=='kmeans'):
        pipeline = Pipeline([('scaler', StandardScaler()), ('pca', PCA()), ('kmeans',kmeans)])
    elif (algorithm=='hierarchical'):
        pipeline = Pipeline([('scaler', StandardScaler()), ('pca', PCA()), ('hierarchical',hierarch)])
    else:
        raise ValueException("unsupported algorithm {}".format(algorithm))

    #x = patchstats
    x = patchstats[:,1:3]  # only mean, std and range
    y = pipeline.fit_predict(x)

    dim = (int)(math.sqrt(y.shape[0]))
    img_clust = np.reshape(y, (dim, dim))

    if show_results:
        print("Visualizing results...")
        plot_with_overlay(img, img_clust, title='cluster heatmap')
        plt.show()

    if show_diagnostics:
        print("Showing diagnostic images...")
        plot_with_overlay(img, img_clust, show_overlay=False, title='original image')
        plot_with_overlay(img, img_clust, show_org=False, title='local clusters')
        plt.show()

    if show_diagnostics_extra:
        print("Showing diagnostic feature images...")
        img_mean = np.reshape(patchstats[:,0], (dim, dim))
        img_median = np.reshape(patchstats[:,1], (dim, dim))
        img_std = np.reshape(patchstats[:,2], (dim, dim))
        img_range = np.reshape(patchstats[:,3], (dim, dim))
        plot_with_overlay(img, img_mean, title='local mean')
        plot_with_overlay(img, img_median, title='local median')
        plot_with_overlay(img, img_std, title='local standard deviation')
        plot_with_overlay(img, img_range, title='local range')

    if return_cluster_image:
        return img, img_clust
```

In [413]:
```
def plot_with_overlay(orgimg, overlayimg, fig_size=(10,10), show_org=True, show_overlay=True,
                      overlay_alpha=0.3, cmapname='Spectral', title=None):
    l = (orgimg.shape[0] - overlayimg.shape[0])
    t = (orgimg.shape[1] - overlayimg.shape[1])
    r = (orgimg.shape[0] - l)
    b = (orgimg.shape[1] - t)

    cmin = 1.1*np.min(overlayimg)
    cmax = 1.1*np.max(overlayimg)
    _ = plt.figure(figsize=fig_size)
    if show_org:
        plt.imshow(orgimg, cmap='gray', origin='upper', extent=[0,orgimg.shape[0], 0, orgimg.shape[1]])
        plt.axis('off')
    else:
        overlay_alpha=0.8
    if (show_overlay):
        plt.imshow(overlayimg, cmap=cmapname, alpha=overlay_alpha, vmin=cmin, vmax=cmax, origin='upper', extent=[l,r,t,b])
        plt.axis('off')
    if (title): plt.title(title)
    plt.show()
```

In [237]:
```
def get_single_cluster_image(cluster_img, cluster_num):
    return (cluster_img==cluster_num).astype(int)

def show_cluster_images(img, cluster_img, n_clusters, show_img=False, cmapname='tab10'):
    for i in range(n_clusters):
        img_clust_i = get_single_cluster_image(cluster_img, i)
        plot_with_overlay(img, img_clust_i, title='cluster {}'.format(i), show_org=show_img, cmapname=cmapname)

def show_single_cluster_image(img, cluster_img, cluster_to_show, show_img=True, opacity=0.5, cmapname='RdYlGn'):
    img_clust_i = get_single_cluster_image(cluster_img, cluster_to_show)
    plot_with_overlay(img, img_clust_i, title='cluster {}'.format(cluster_to_show), show_org=show_img, overlay_alpha=opacity, cmapname=cmapname)
```

Look at the data

In [397]:
```
n_patches_y = 10
n_patches_x = 10
df = imgutils.slicestats(imgfiles, n_patches_y, n_patches_x, feature_funcs)
```

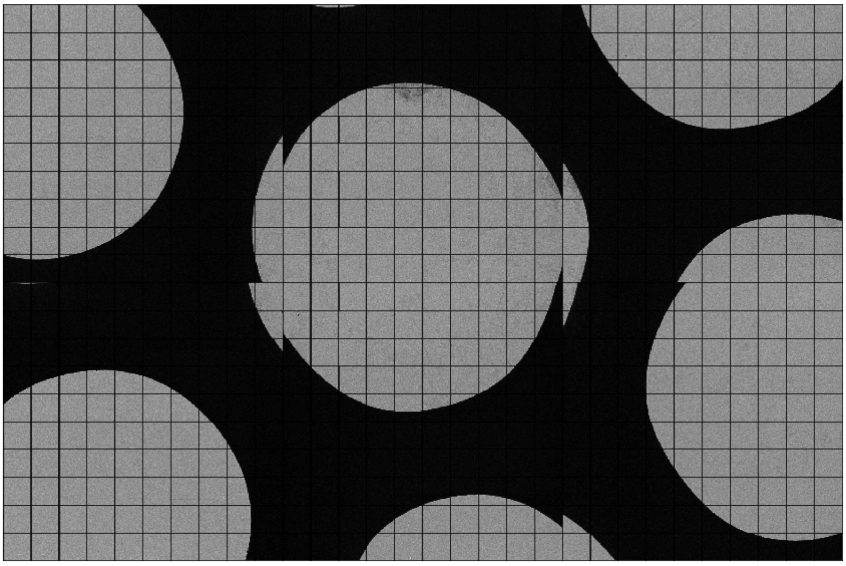
In [398]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.7566955672767465
Hierarchical:  0.5896559559617408
```

In [399]:
```
df['dummy'] = 0
imgutils.show_large_heatmap(df, 'dummy', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,10))
```

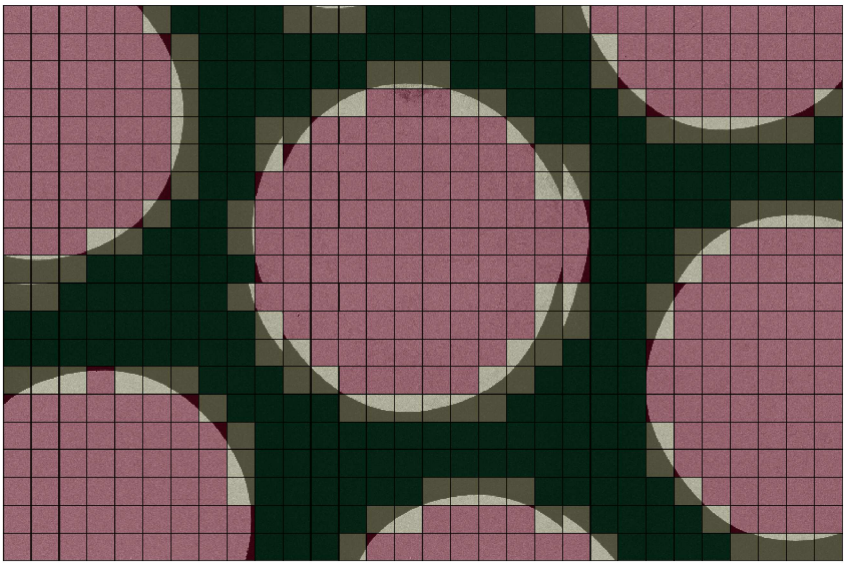
Heats from: dummy



In [404]:
```
swap_clusters(df, 'kmeans', 1,2)
```

In [405]:
```
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,10))
```

Heats from: kmeans



Try on single image with much smaller patch size

In [401]:
```
n_patches = 40
df2 = imgutils.slicestats([imgfile1], n_patches, n_patches, feature_funcs)
```

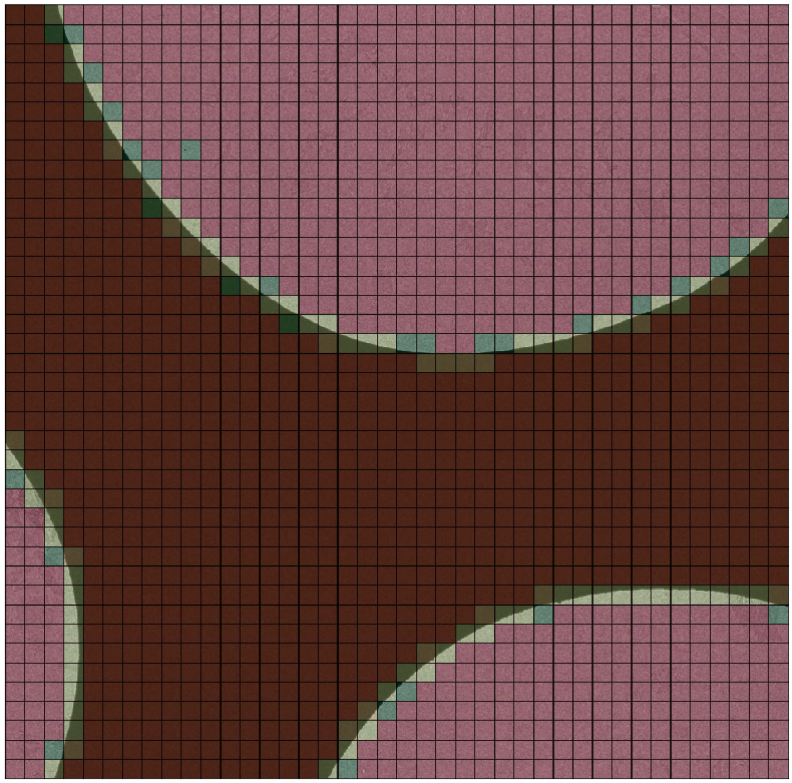
In [402]:
```
run_ml_pipelines(df2, feature_names, 6, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.8847695277863312
Hierarchical:  0.6840381187506879
```

In [403]:
```
imgutils.show_large_heatmap(df2, 'kmeans', [imgfile1], n_rows=1, n_cols=1, fig_size=(14,14))
```

Heats from: kmeans



This set requires the 2 step approach (i.e. first filter out the black tiles)

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)

Try two-step pipeline - first on single tile

step 1: filter out black tiles

step 2: cluster remaining tiles

Parametrize:

In [200]:
```
n_clusters_step1 = 2
n_clusters_step2_kmeans = 3
n_clusters_step2_hierarchical = 3

n_patches_x = 40
n_patches_y = 40

n_tiles_x = 1
n_tiles_y = 1
```

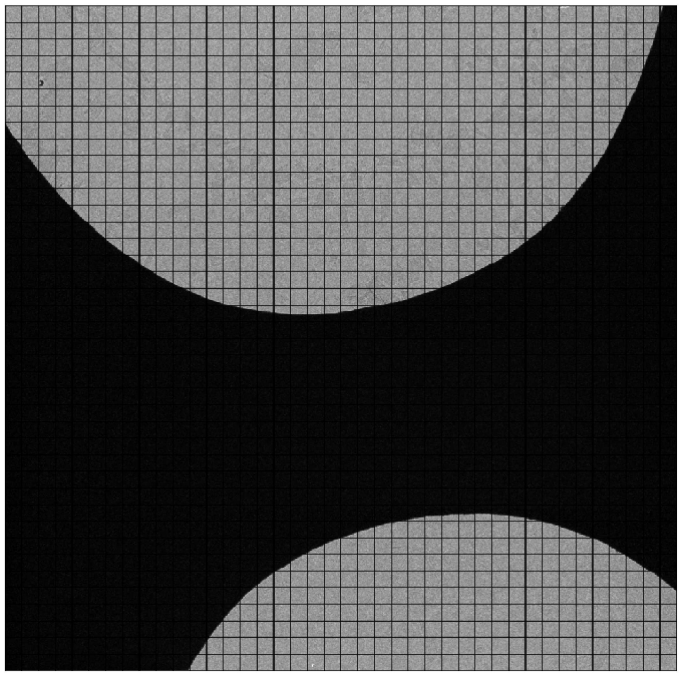
Feature extract

In [201]:
```
# reset
#df = df.drop(columns=['kmeans'])
df2 = None
df3 = None
```

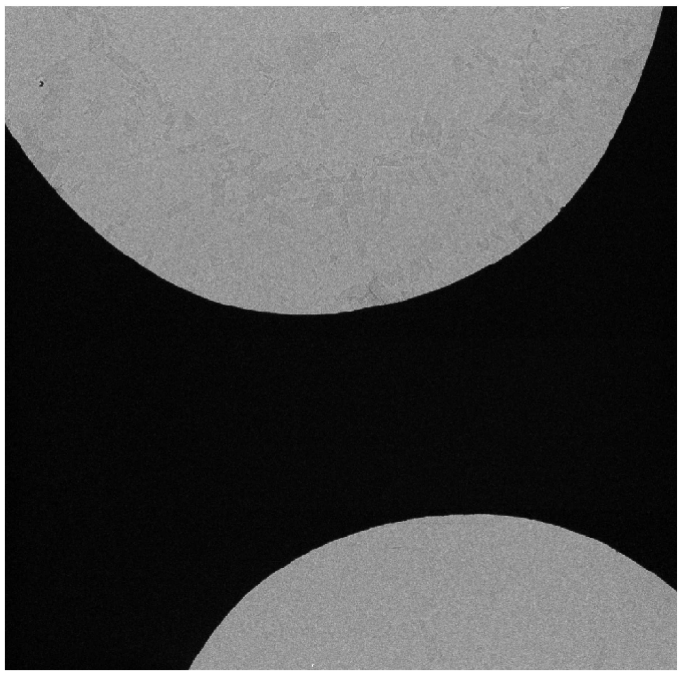
In [202]:
```
imgfiles = import_data(datafolder)
imgfiles = [imgfiles[4]]
df, feature_names = extract_features(imgfiles, feature_funcs, n_patches_y, n_patches_x)
```

In [203]:
```
df['dummy']=0
imgutils.show_large_heatmap(df, 'dummy', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,12))
imgutils.show_large_heatmap(df, 'dummy', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,12), no_borders=True)
```

Heats from: dummy



Heats from: dummy



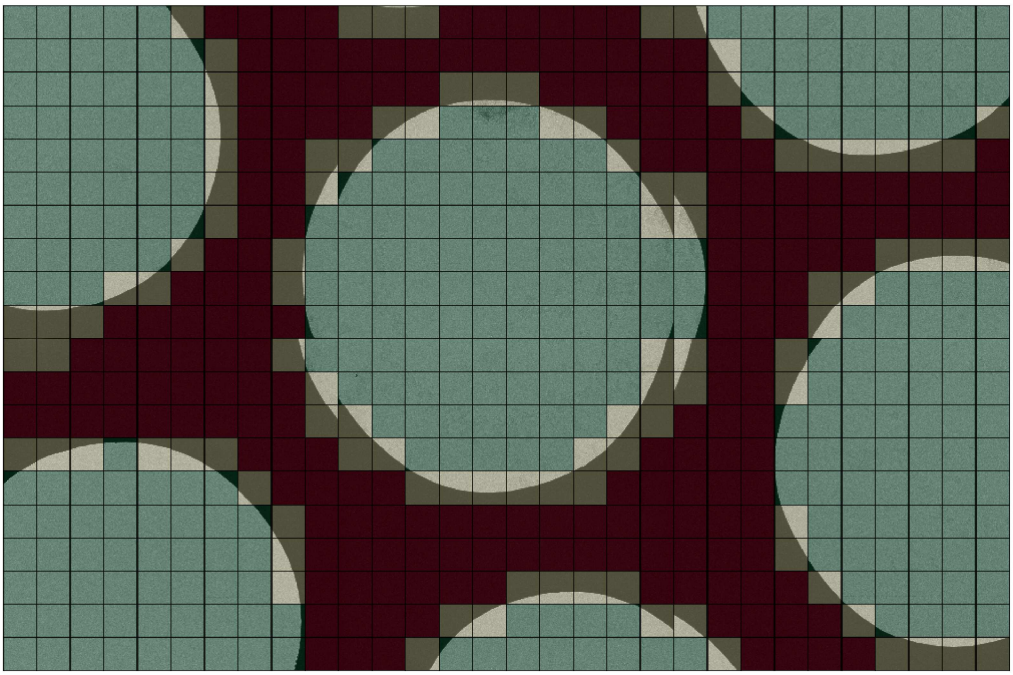
Step 1: filter-out black tiles

In [204]:
```
_ = run_kmeans_pipeline(df, feature_names, n_clusters_step1, standardize=True, use_pca=True )
```

In [435]:
```
swap_clusters(df,'kmeans', 1, 2)
```

In [436]:
```
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(12,12))
```

Heats from: kmeans



In [179]:
```
df['kmeans'].value_counts()
```

Out[179]:
```
0    811
2    780
1      9
Name: kmeans, dtype: int64
```

In [180]:
```
# 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)
```

```
0
```

In [181]:
```
# NO, it's cat 2!
cat_select = 2
```

In [182]:
```
df2 = df[df['kmeans']==cat_select]
df2.head(3)
```

Out[182]:

	filename	s_y	s_x	n_y	n_x	alias	img_mean	img_std	img_median	img_mode	img_kurtosis	img_skewness	dummy	kmeans
0	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	0	40	40	img0_0-0	639.0352	39.471047	637.0	643.798340	-0.216597	0.200653	0	2
1	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	1	40	40	img0_0-1	632.6768	39.445166	632.0	644.831543	-0.022774	0.106714	0	2
2	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	2	40	40	img0_0-2	627.9840	41.394219	626.0	616.825684	0.316601	0.099798	0	2

In [183]:
```
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 [184]:
```
df2['kmeans'].value_counts()
```

Out[184]:
```
0    726
2     38
1     16
Name: kmeans, dtype: int64
```

In [185]:
```
df2['hierarchical'].value_counts()
```

Out[185]:
```
0    764
1     13
2      3
Name: hierarchical, dtype: int64
```

In [186]:
```
df2=df2.rename(columns = {'kmeans':'kmeans2', 'hierarchical':'hierarchical2'})
```

In [187]:
```
df2.head(3)
```

Out[187]:

	filename	s_y	s_x	n_y	n_x	alias	img_mean	img_std	img_median	img_mode	img_kurtosis	img_skewness	dummy	kmeans2	hierarchica
0	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	0	40	40	img0_0-0	639.0352	39.471047	637.0	643.798340	-0.216597	0.200653	0	0	0
1	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	1	40	40	img0_0-1	632.6768	39.445166	632.0	644.831543	-0.022774	0.106714	0	0	0
2	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	2	40	40	img0_0-2	627.9840	41.394219	626.0	616.825684	0.316601	0.099798	0	0	0

In [188]:
```
df3 = df.merge(df2, 'left')
```

In [189]:
```
df3.head(3)
```

Out[189]:

	filename	s_y	s_x	n_y	n_x	alias	img_mean	img_std	img_median	img_mode	img_kurtosis	img_skewness	dummy	kmeans	kmeans2	h
0	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	0	40	40	img0_0-0	639.0352	39.471047	637.0	643.798340	-0.216597	0.200653	0	2	0.0	0
1	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	1	40	40	img0_0-1	632.6768	39.445166	632.0	644.831543	-0.022774	0.106714	0	2	0.0	0
2	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	2	40	40	img0_0-2	627.9840	41.394219	626.0	616.825684	0.316601	0.099798	0	2	0.0	0

In [190]:
```
df3['kmeans2'].fillna(value=-1, inplace=True)
```

In [191]:
```
df3['hierarchical2'].fillna(value=-1, inplace=True)
```

In [192]:
```
df3.head(3)
```

Out[192]:

	filename	s_y	s_x	n_y	n_x	alias	img_mean	img_std	img_median	img_mode	img_kurtosis	img_skewness	dummy	kmeans	kmeans2	h
0	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	0	40	40	img0_0-0	639.0352	39.471047	637.0	643.798340	-0.216597	0.200653	0	2	0.0	0
1	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	1	40	40	img0_0-1	632.6768	39.445166	632.0	644.831543	-0.022774	0.106714	0	2	0.0	0
2	..\data\Polymers_27Sep2018\LowMag_1K_Subset\Ti...	0	2	40	40	img0_0-2	627.9840	41.394219	626.0	616.825684	0.316601	0.099798	0	2	0.0	0

In [193]:
```
df3['heats']=df3['kmeans2']+1
```

In [194]:
```
df3['heats'].value_counts()
```

Out[194]:
```
0.0    820
1.0    726
3.0     38
2.0     16
Name: heats, dtype: int64
```

In [195]:
```
# make the whole 2 clusters only
#df3['heats'].replace({1:0}, inplace=True)
#df3['heats'].value_counts()
```

In [196]:
```
df3['heats2']=df3['hierarchical2']+1
```

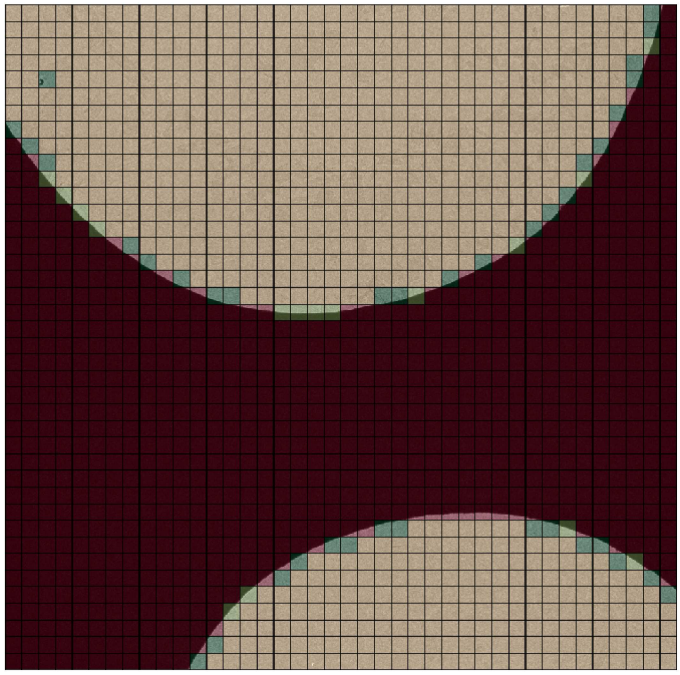
In [197]:
```
df3['heats2'].value_counts()
```

Out[197]:
```
0.0    820
1.0    764
2.0     13
3.0      3
Name: heats2, dtype: int64
```

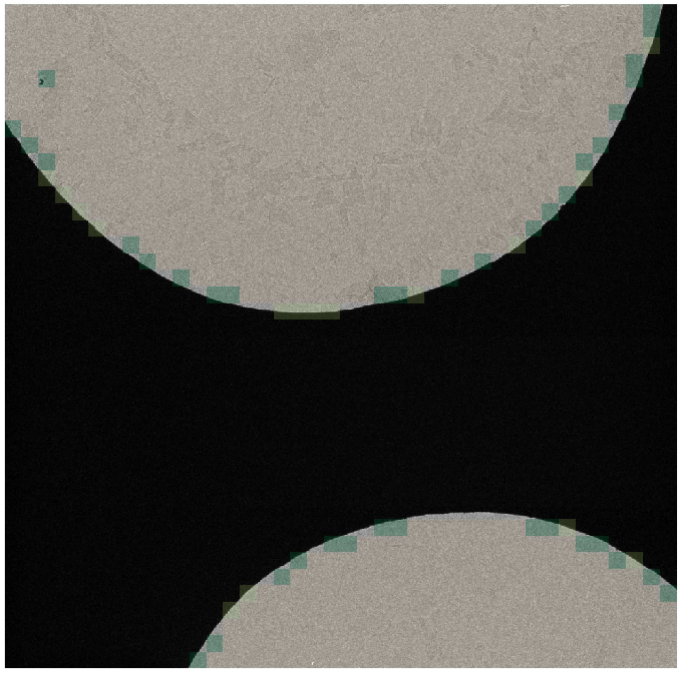
In [198]:
```
# make the whole 2 clusters only
#df3['heats2'].replace({1:0}, inplace=True)
#df3['heats2'].value_counts()
```

In [199]:
```
imgutils.show_large_heatmap(df3, 'heats', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(16,12))
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



Heats from: heats



Try sanitized version

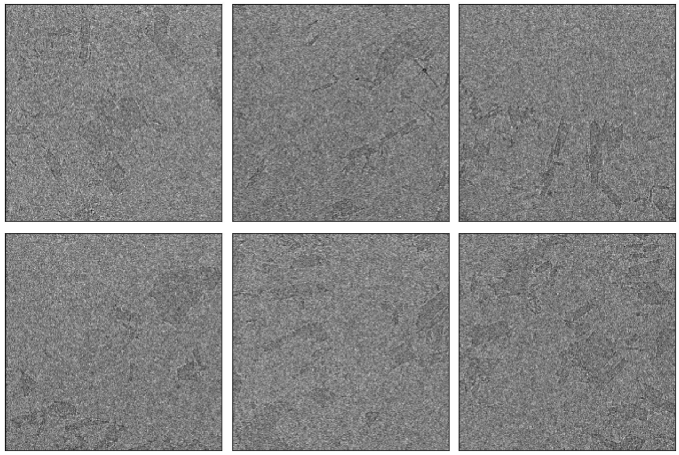
In [367]:
```
datafolder_s = '../data/Polymers_27Sep2018/LowMag_NoBlack'
n_tiles_x_s = 3  # mostly for visualization
n_tiles_y_s = 2

n_patches_y_s=10
n_patches_x_s=10
```

In [368]:
```
feature_funcs2 = feature_funcs
feature_names2 = imgutils.stat_names(feature_funcs2)
```

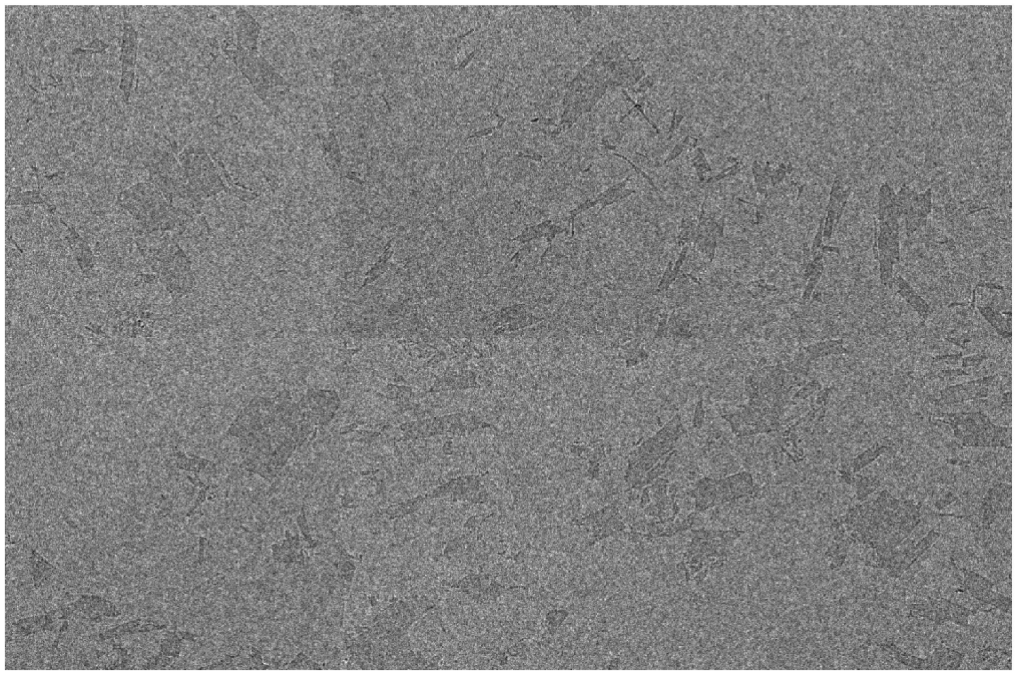
In [369]:
```
imgfiles_s = import_data(datafolder_s)
imgfiles_s = imgfiles_s[:6]
df_s, feature_names = extract_features(imgfiles_s, feature_funcs2, n_patches_y_s, n_patches_x_s)
```

In [376]:
```
imgutils.showimgset(imgfiles_s, 2,3, fig_size=(12, 8), relspacing=(0.05,0.05))
```



In [370]:
```
df_s['dummy']=0
imgutils.show_large_heatmap(df_s, 'dummy', imgfiles_s, n_rows=n_tiles_y_s, n_cols=n_tiles_x_s, fig_size=(12,12), no_borders=True)
```

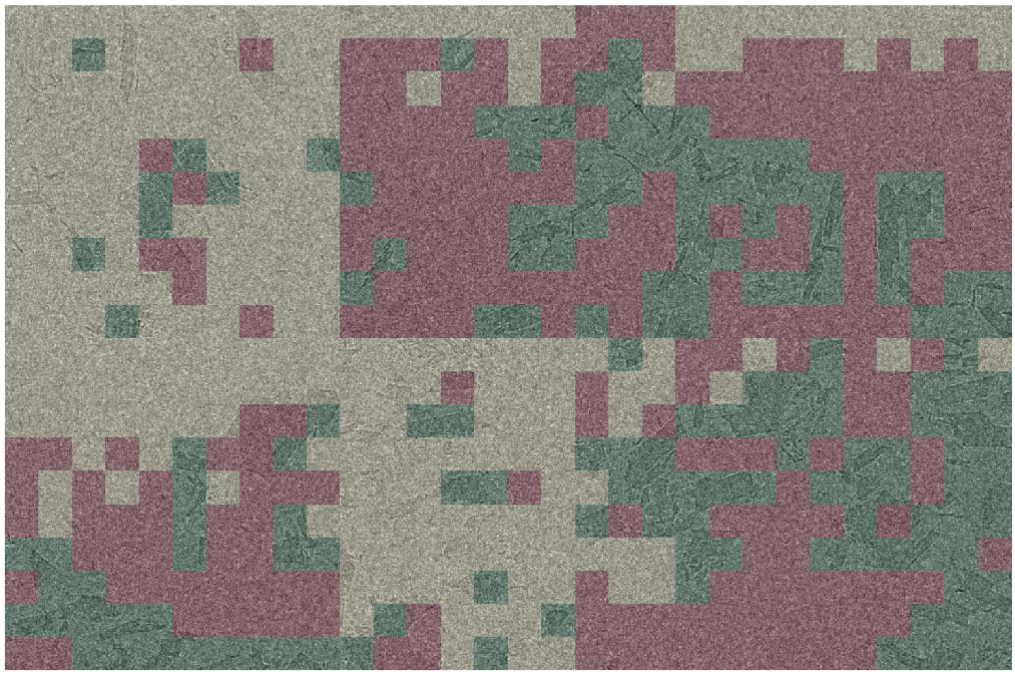
Heats from: dummy



In [371]:
```
_ = run_kmeans_pipeline(df_s, feature_names2, 3, standardize=True, use_pca=True )
```

In [372]:
```
imgutils.show_large_heatmap(df_s, 'kmeans', imgfiles_s, n_rows=n_tiles_y_s, n_cols=n_tiles_x_s, fig_size=(12,12), opacity=0.2, no_borders=True)
```

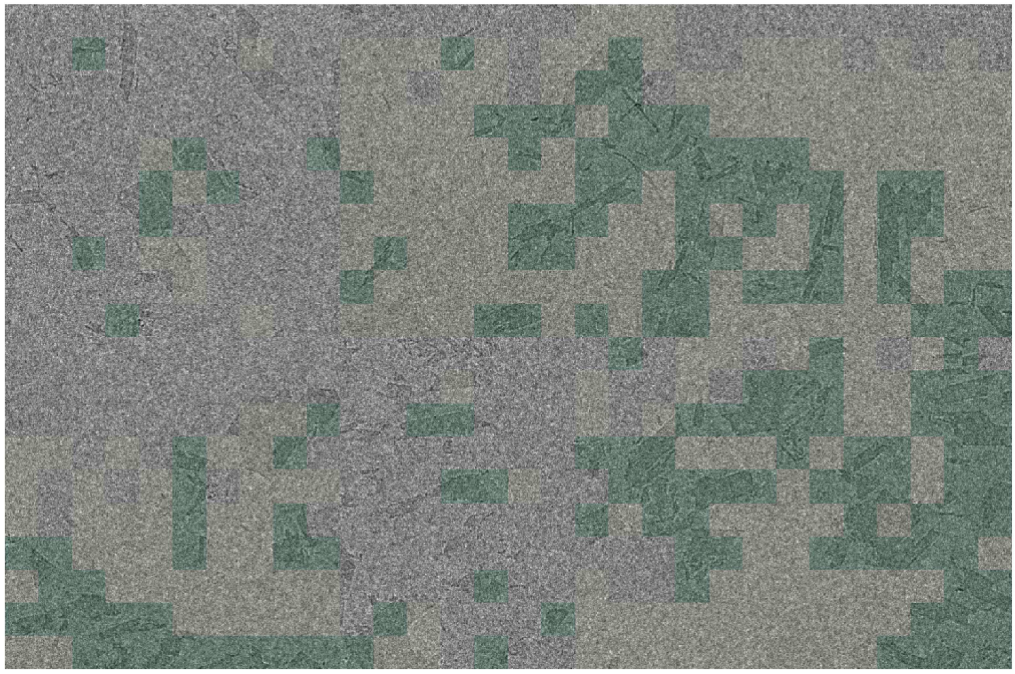
Heats from: kmeans



In [373]:
```
swap_clusters(df_s, 'kmeans', 0,1)
```

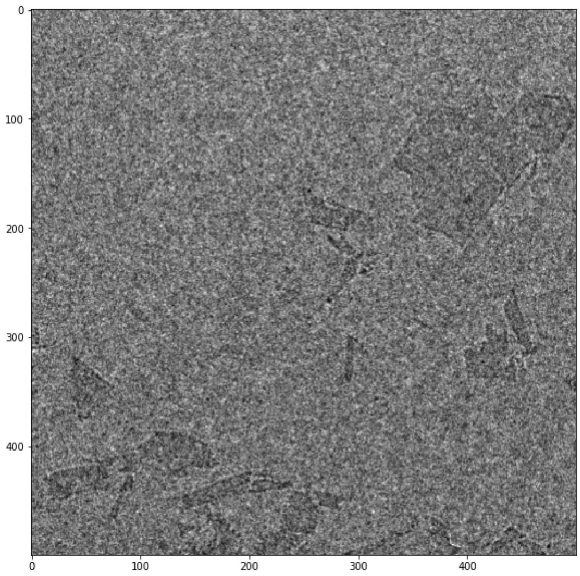
In [374]:
```
imgutils.show_large_heatmap(df_s, 'kmeans', imgfiles_s, n_rows=n_tiles_y_s, n_cols=n_tiles_x_s, fig_size=(12,12), opacity=0.2, no_borders=True, heatdependent_opacity=True)
```

Heats from: kmeans



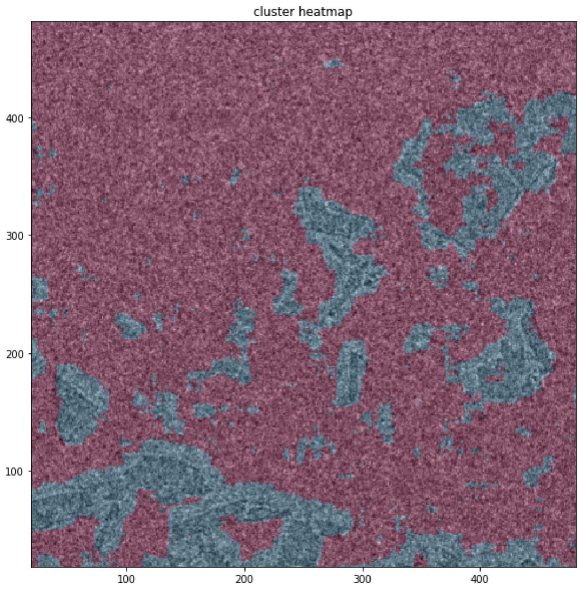
Try sliding window

In [249]:
```
imgfilename = imgfiles_s[3]
img = imgutils.loadtiff(imgfilename)
imgutils.showimg(img, fig_size=(10,10))
```



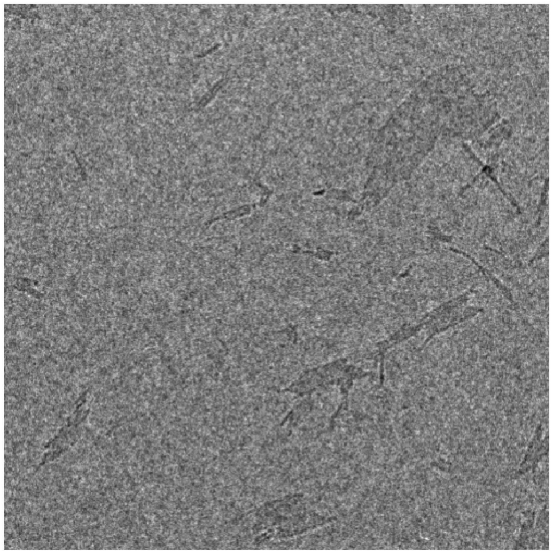
In [255]:
```
img, h = run_pipeline_windowed(imgfilename, (20,20), 2, return_cluster_image=True, downscale_factor=1)
```

```
Importing image(s)s...
Extracting features... 100 %
Clustering into 2 clusters...
Visualizing results...
```



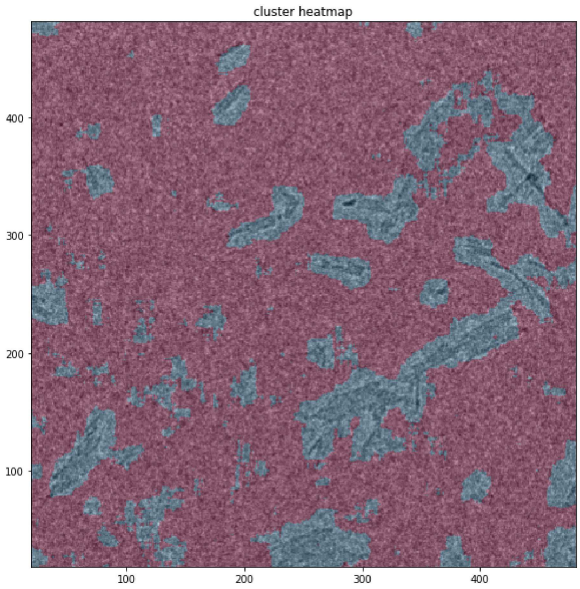
and on another

In [417]:
```
imgfilename = imgfiles_s[1]
img = imgutils.loadtiff(imgfilename)
imgutils.showimg(img, fig_size=(10,10))
```

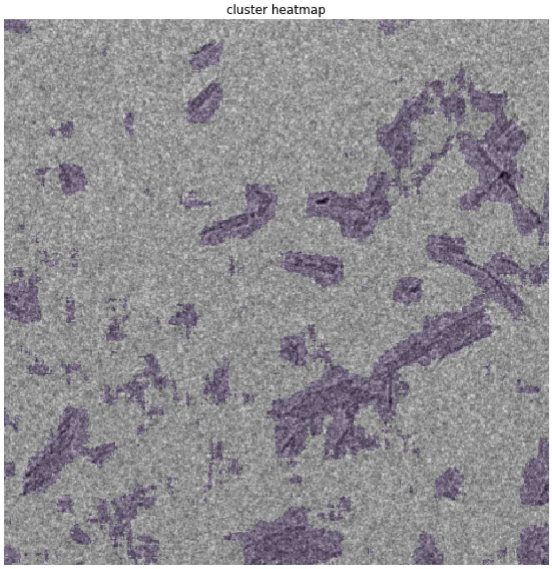


In [410]:
```
img, h = run_pipeline_windowed(imgfilename, (20,20), 2, return_cluster_image=True, downscale_factor=1)
```

```
Importing image(s)s...
Extracting features... 100 %
Clustering into 2 clusters...
Visualizing results...
```



In [433]:
```
plot_with_overlay(img, h, title='cluster heatmap', cmapname='Purples', fig_size=(10,10), overlay_alpha=0.2)
```



In [434]:
```
show_cluster_images(img, h, 2, show_img=False)
```

cluster 0

cluster 1