Full Pipeline (on Tileset7) - Aug 2017

Created: 21 Aug 2018
Last update: 29 Aug 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 [ ]:
```
# 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 [2]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)
```

---

2. Data Definitions & Feature Specification

In [3]:
```
# 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 [4]:
```
# 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/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 [5]:
```
# data hyper-parameters
default_n_clusters = 3

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

In [6]:
```
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 [9]:
```
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_p
ca=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 [10]:
```
run_ml_pipelines2(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.3834680571101721
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 [34]:
```
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 [12]:
```
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_pc
a=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 [13]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=True)
```

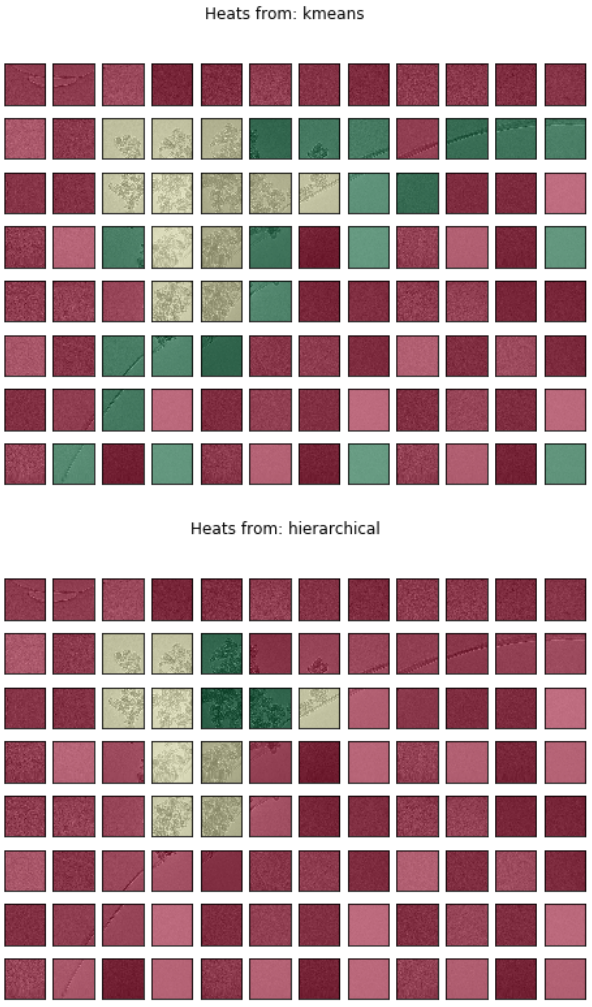
```
Executing clustering pipelines...
Done


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

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 [15]:
```
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:  0.5272951661092292
Hierarchical:  0.5543189694833234
```

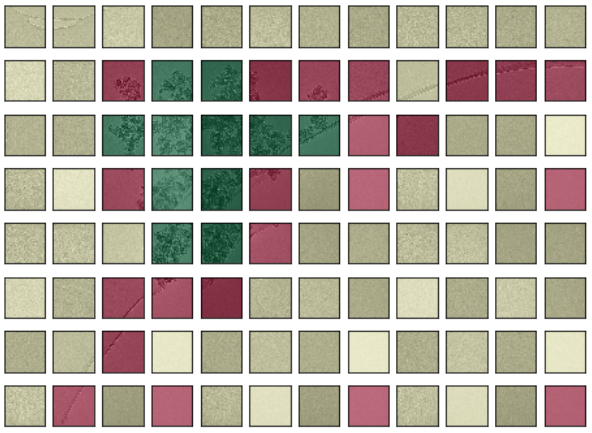


Run it again without PCA and/pr normalization compare results

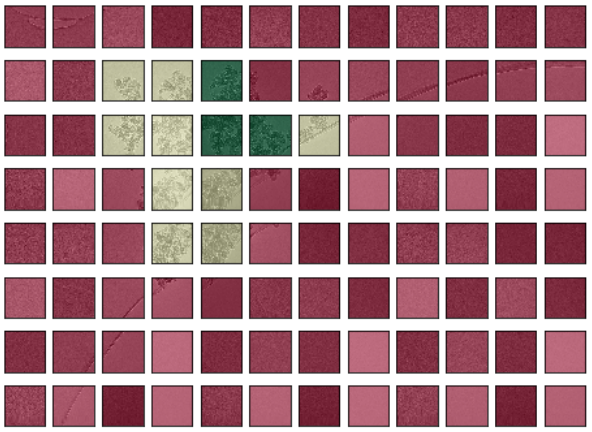
In [16]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=True, use_pca=False)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.5251916981567151
Hierarchical:  0.5543189694833234
```

Heats from: kmeans



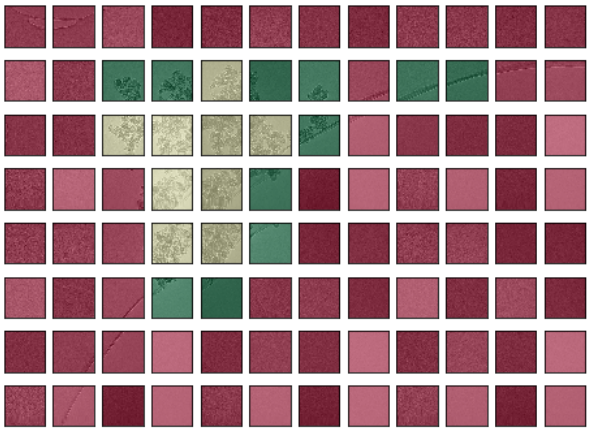
Heats from: hierarchical



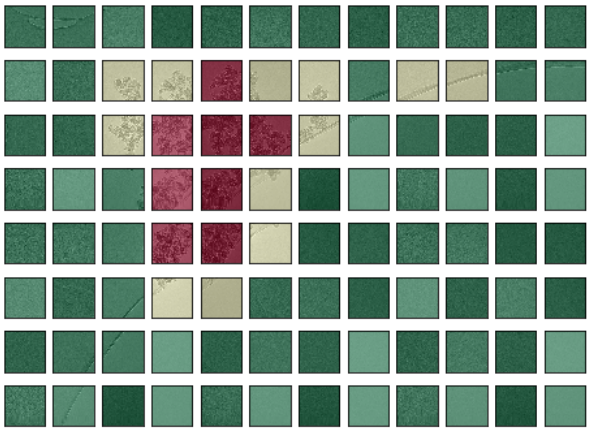
In [17]:
```
run_ml_pipelines(df, feature_names, default_n_clusters, standardize=False, use_pca=False)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

```
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 [18]:
```
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 [19]:
```
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 [20]:
```
def run_fullpipeline(imagefolder, n_image_rows, n_image_cols,
                     n_grid_rows, n_grid_cols, feature_funcs, n_clusters):
    """
    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=s)
    imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_image_rows, n_cols=n_image_cols, fig_size=s)
```

In [ ]:
```
```

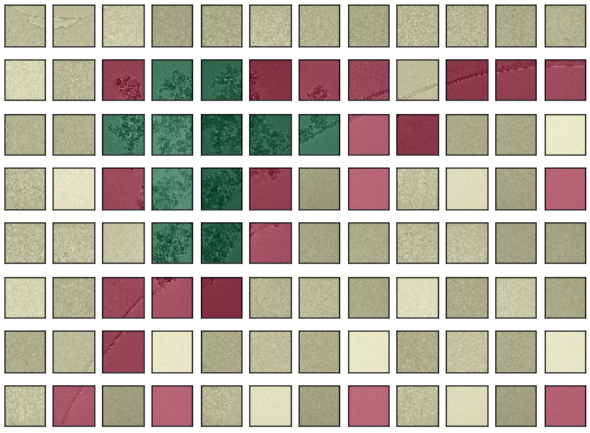
7. Try it out with different combinations of slices

4x4 - 3 clusters

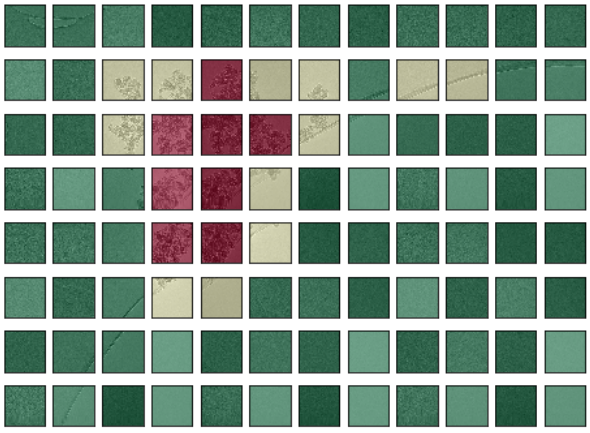
In [21]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 3)
```

```
Working...
Results:
Score k-means: 0.5251916981567152
Score hierarchical: 0.6138055241918071
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical

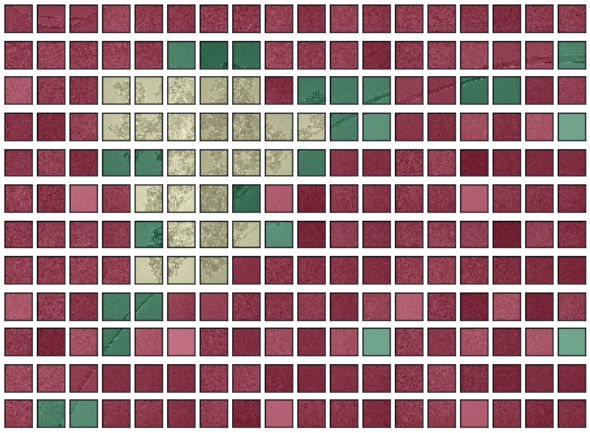


6x6, 3 clusters

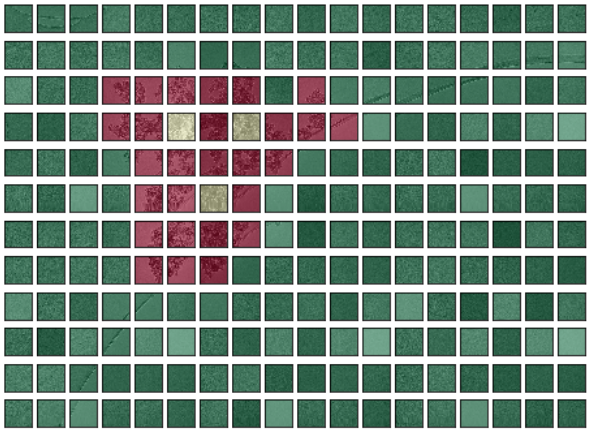
In [22]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 3)
```

```
Working...
Results:
Score k-means: 0.6020413797895592
Score hierarchical: 0.7674936226598582
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical

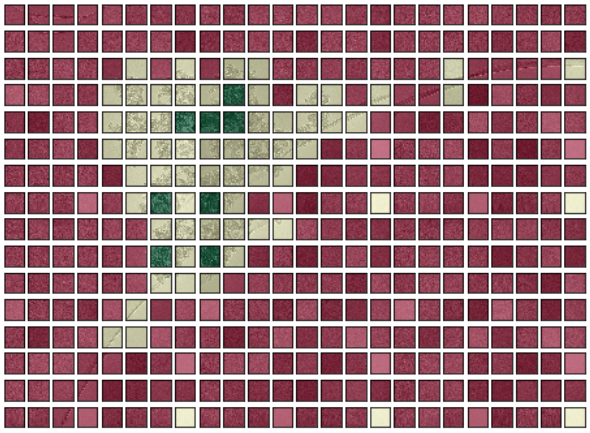


8x8, 3 clusters

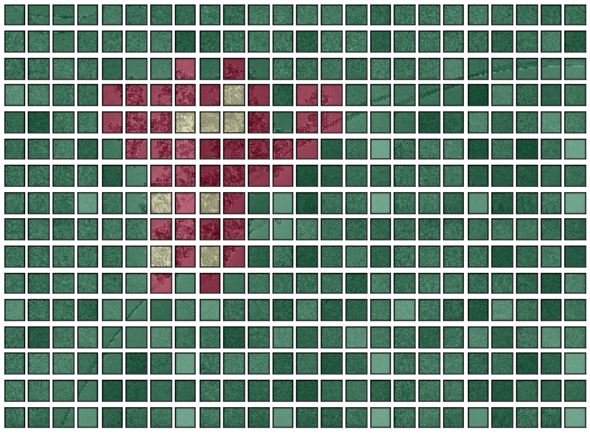
In [23]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 3)
```

```
Working...
Results:
Score k-means: 0.6637853349132272
Score hierarchical: 0.7699361876771397
Visualizing...
```

Heats from: kmeans



Heats from: hierarchical



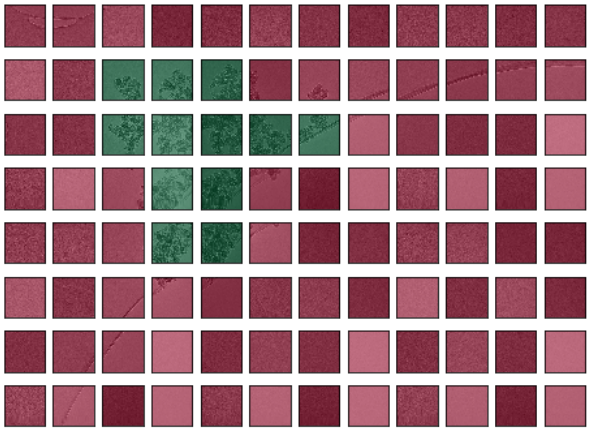
8. Try it out with different number of clusters

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

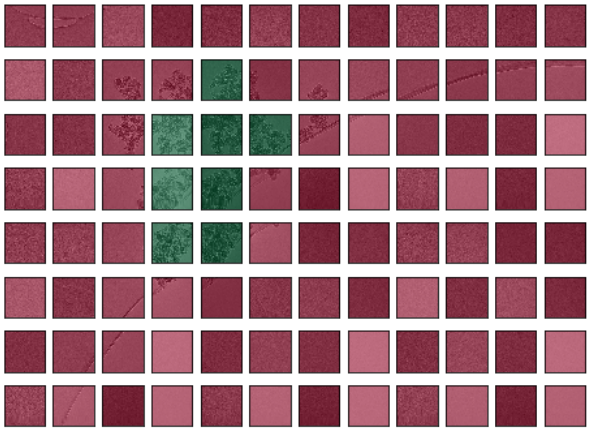
In [24]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 2)
```

```
Working...
Results:
Score k-means: 0.5886428860898483
Score hierarchical: 0.7507303799260893
Visualizing...
```

Heats from: kmeans



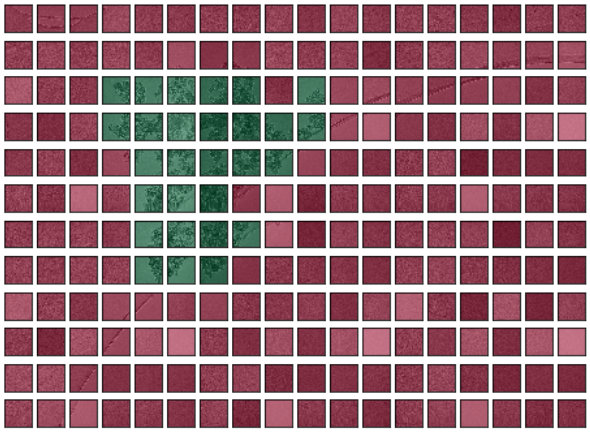
Heats from: hierarchical



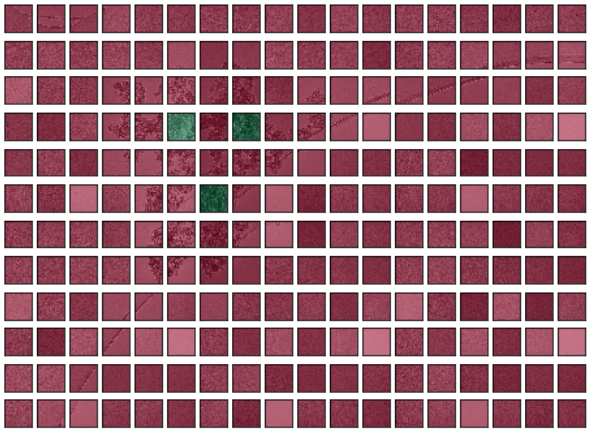
In [25]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 2)
```

```
Working...
Results:
Score k-means: 0.633254261385902
Score hierarchical: 0.7959516863373638
Visualizing...
```

Heats from: kmeans



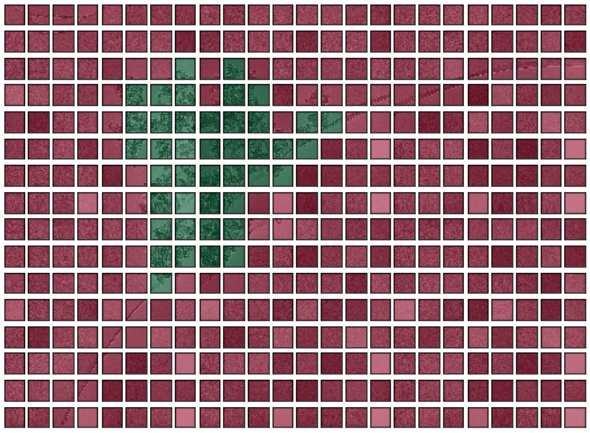
Heats from: hierarchical



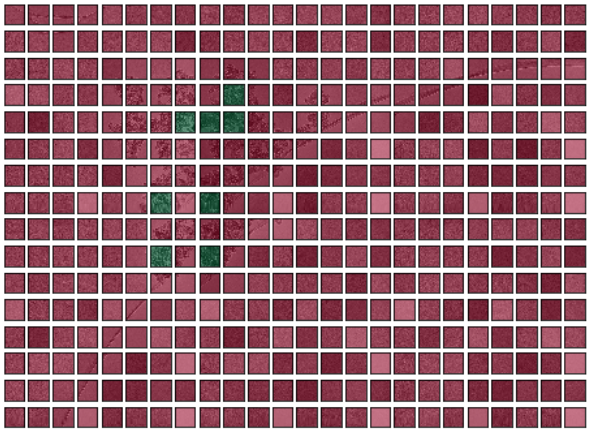
In [26]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 8, 8, feature_funcs, 2)
```

```
Working...
Results:
Score k-means: 0.6695154011956168
Score hierarchical: 0.8396991318389545
Visualizing...
```

Heats from: kmeans



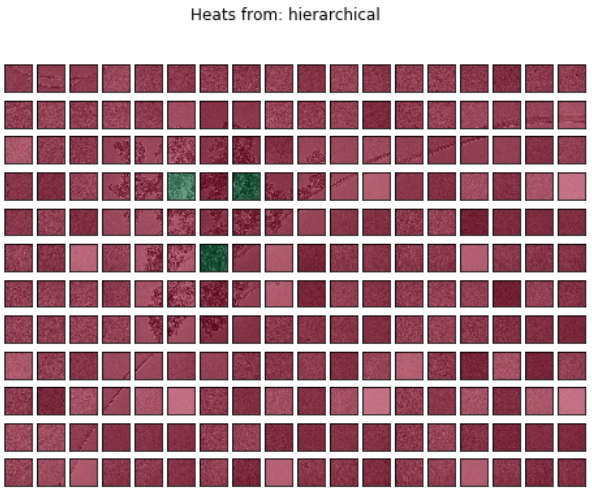
Heats from: hierarchical



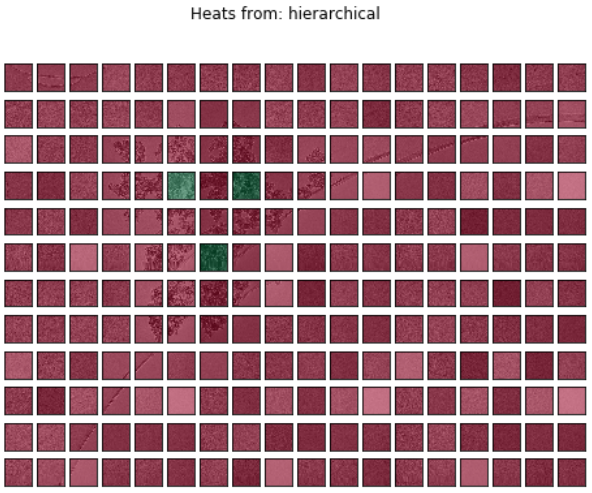
Look again at hierarchical with scaling and pca on:

In [35]:
```
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
```

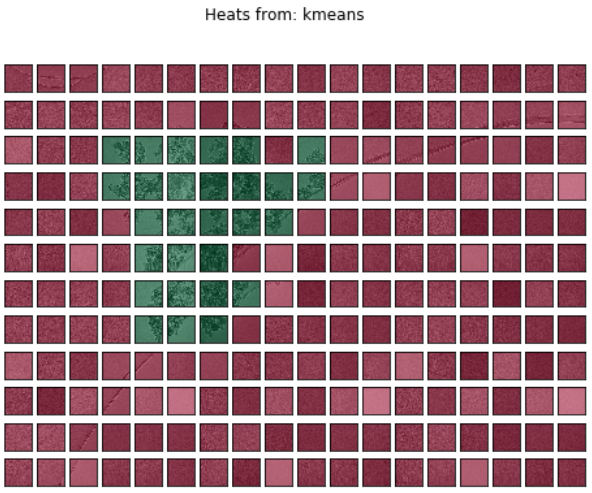



```
Without scaling & pca: 0.7959516863373638
```

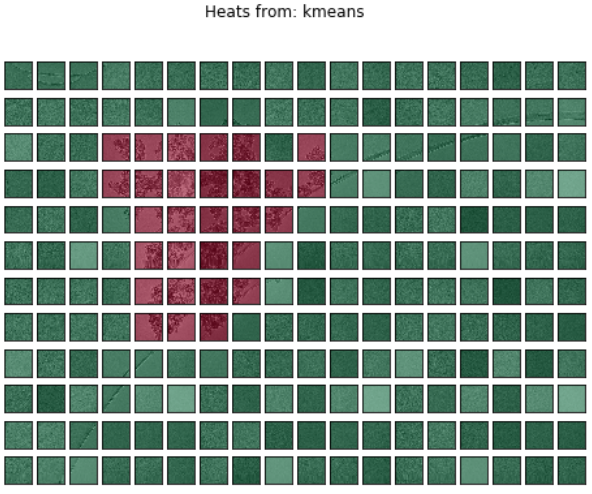


In [36]:
```
 imgfiles = import_data(datafolder)
df_temp, feature_names = extract_features(imgfiles, feature_funcs, 6, 6)
score = run_kmeans_pipeline(df_temp, feature_names, 2, standardize=True, use_pca=True)
print("With scaling & pca:", score)
imgutils.show_large_heatmap(df_temp, 'kmeans', imgfiles, 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
```




```
Without scaling & pca: 0.7638653726133209
```



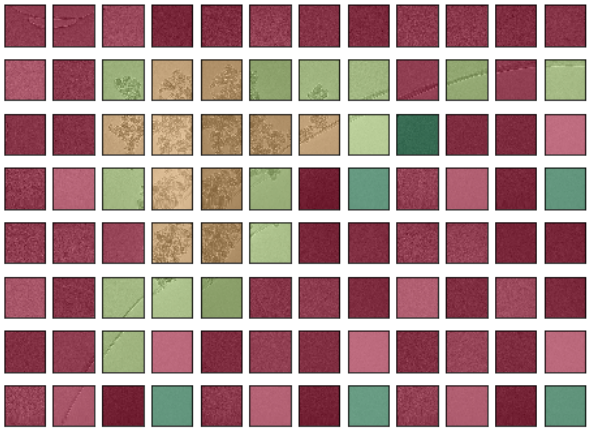
Results with and without scaling / pca are pretty close actually.

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

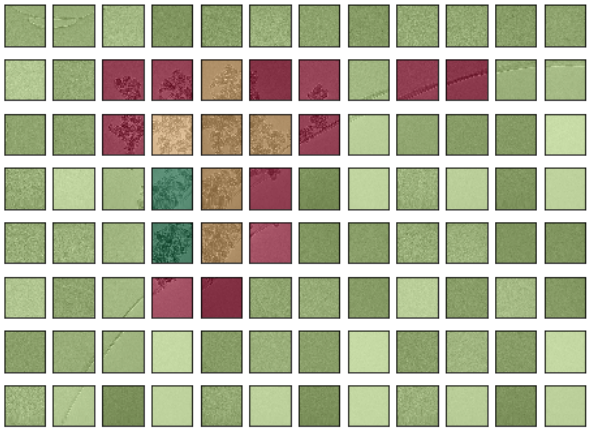
In [37]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 4, 4, feature_funcs, 4)
```

```
Working...
Results:
Score k-means: 0.546591556148677
Score hierarchical: 0.6079930287147731
Visualizing...
```

Heats from: kmeans



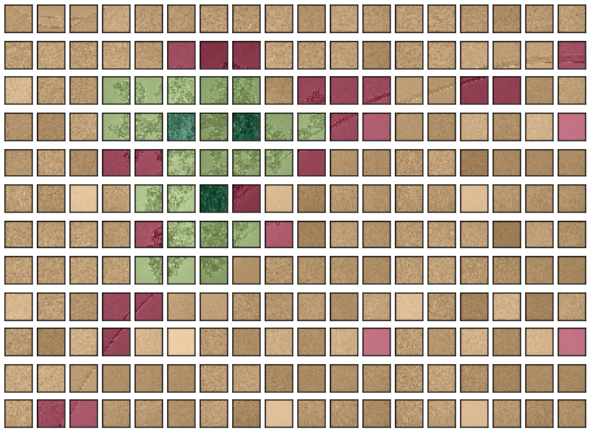
Heats from: hierarchical



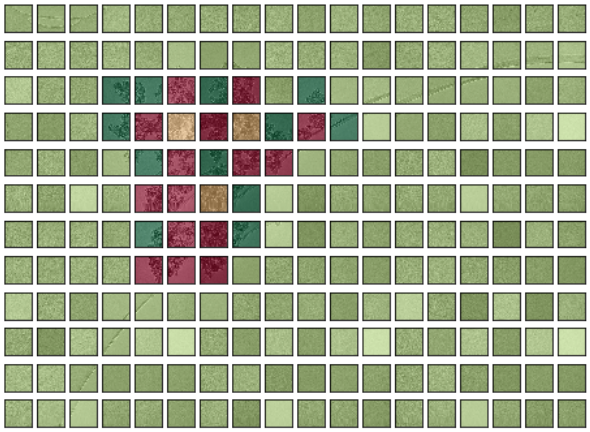
In [38]:
```
run_fullpipeline(datafolder, n_tiles_y, n_tiles_x, 6, 6, feature_funcs, 4)
```

```
Working...
Results:
Score k-means: 0.621537336514108
Score hierarchical: 0.7080629222772171
Visualizing...
```

Heats from: kmeans

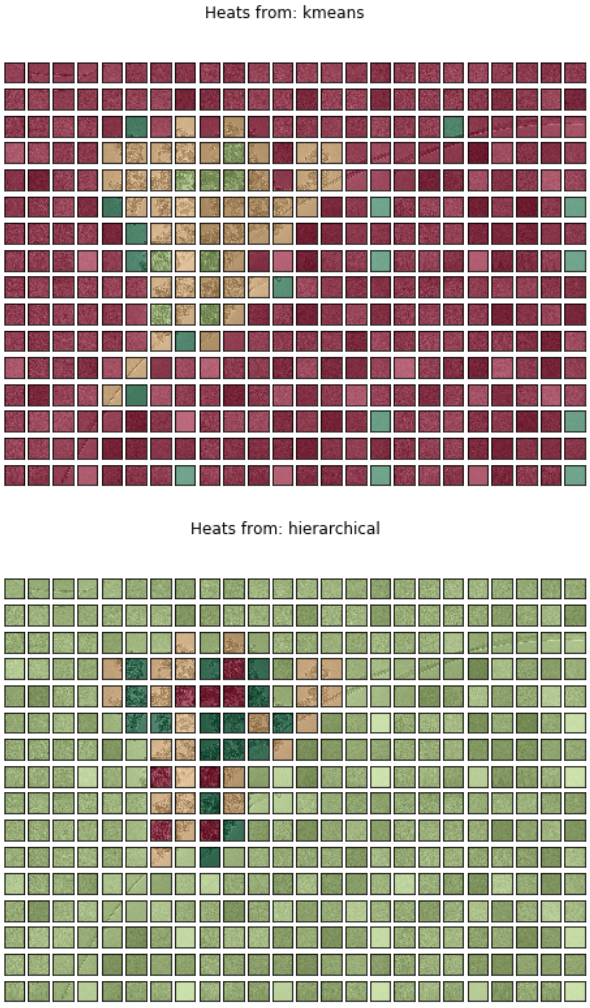


Heats from: hierarchical



In [39]:
```
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...
```



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

9. (Grid) Search for hyper-parameter optimizations

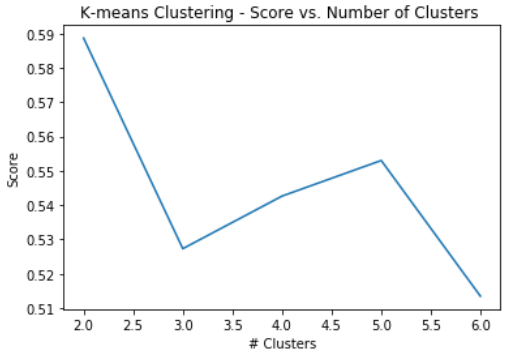
In [42]:
```
imgfiles = import_data(datafolder)
df, feature_names = extract_features(imgfiles, feature_funcs, default_grid_y, default_grid_x)
```

9A. Number of clusters

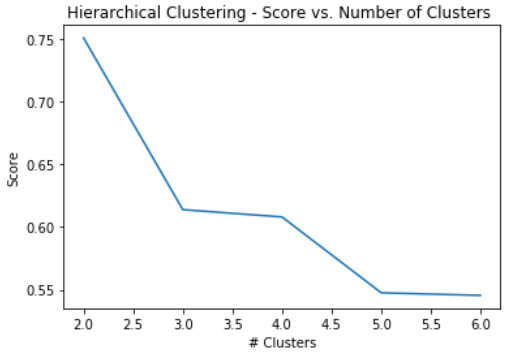
In [46]:
```
def optimize_kmeans_nclusters(df_imgstats, feature_names, n_clusters_start, n_clusters_end):
    """
    Run the full pipeline (except for visualization) for different number of clusters
    and show graph with score vs number of clusters.
    """
    n_count = n_clusters_end + 1 - n_clusters_start
    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 [55]:
```
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 [53]:
```
optimize_kmeans_nclusters(df, feature_names, 2, 6)
```



In [56]:
```
optimize_hierarchical_nclusters(df, feature_names, 2, 6)
```

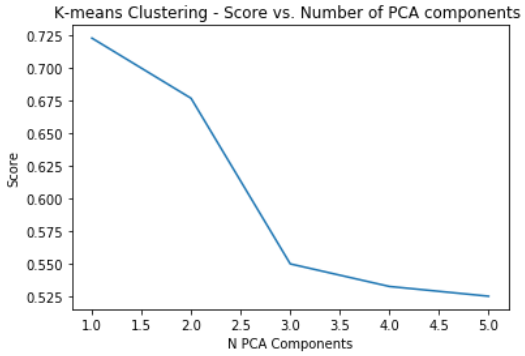


9B. Number of features / PCA Components

In [57]:
```
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()
```

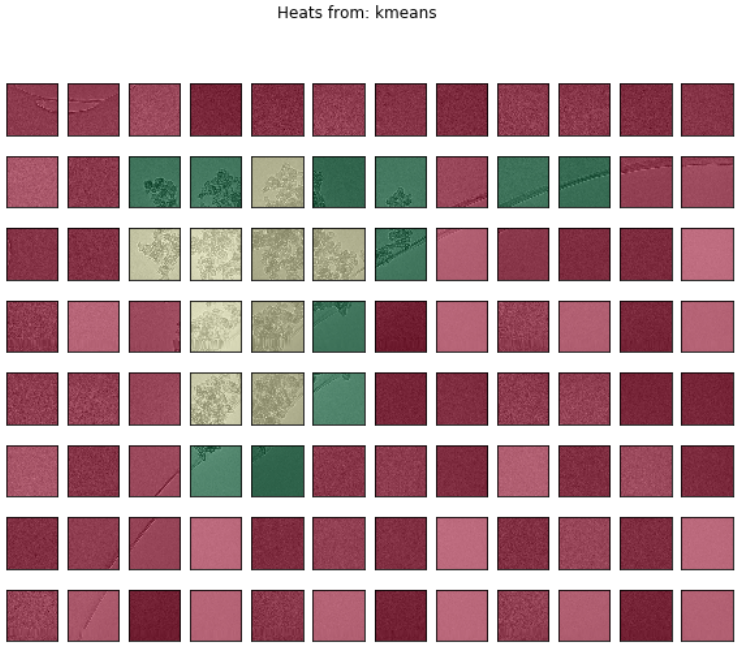
In [58]:
```
def optimize_hierarchical_pcacomponents(df_imgstats, feature_names, n_clusters):
    """
    Run the full pipeline (except for visualization) for different number of PCA componenents
    and show graph with the score vs number of components.
    """
    n_count = len(feature_names) - 1
    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 [59]:
```
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

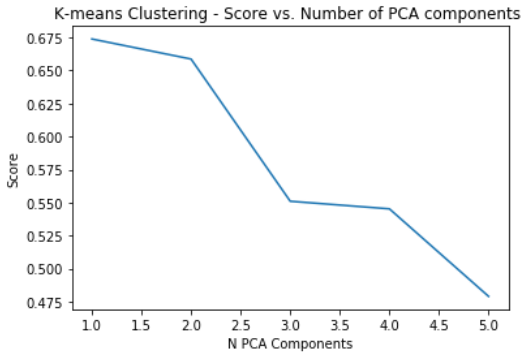
In [60]:
```
score = run_kmeans_pipeline(df, feature_names, 3, standardize=True, use_pca=True, n_pca=1)
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=(10,8))
```



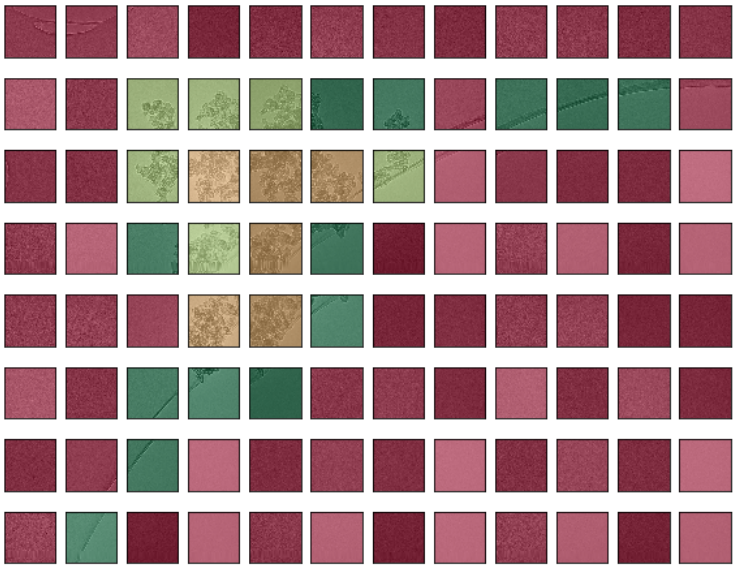
Indeed not bad. Maybe not so surprise as in early EDA we saw that on this particular set the standard deviation is very informative (we used it for manual labelling)

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

In [62]:
```
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))
```



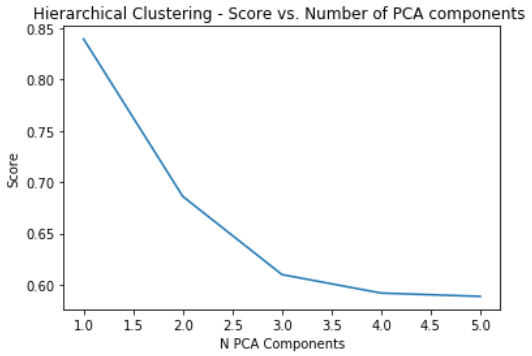
Heats from: kmeans

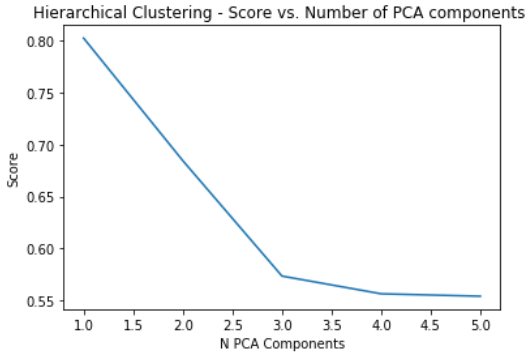


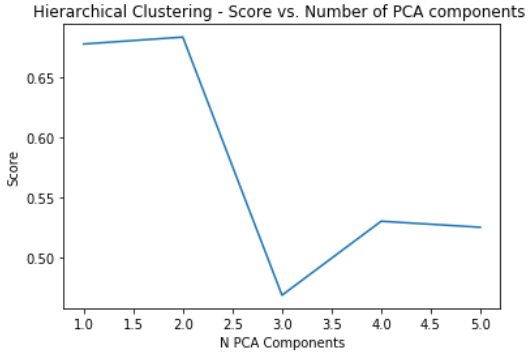
Actually this looks very good! It properly grouped 'fully occupied grid cells' from 'partially occupied'!

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

In [66]:
```
optimize_hierarchical_pcacomponents(df, feature_names, 2)
optimize_hierarchical_pcacomponents(df, feature_names, 3)
optimize_hierarchical_pcacomponents(df, feature_names, 4)
```



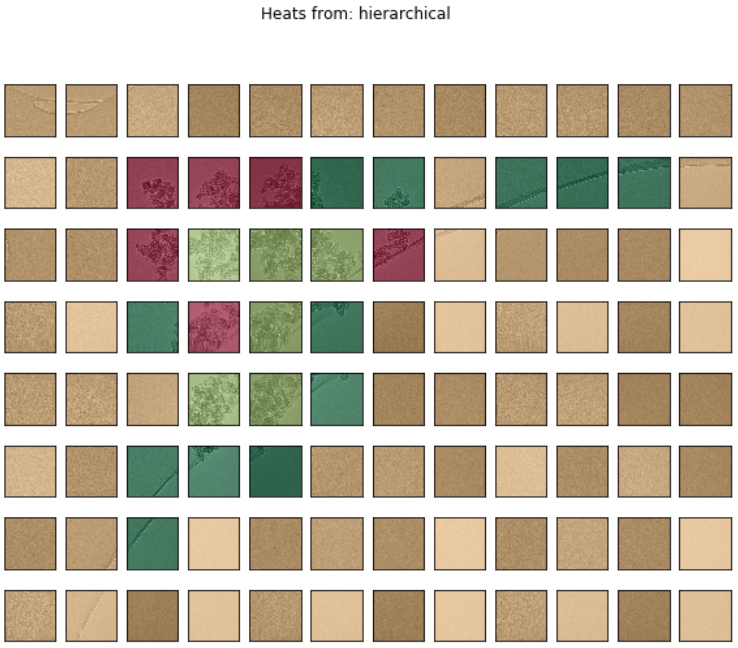




Don't know what these graphs really say; as the score is assessing cluster cohesion based on the last transformation in the pre-processing, it can be expected that less components are more separated. Maybe need to look at the 'elbows' here?

In [69]:
```
# visualize for 4 clusters with 1 pca component:
```

In [68]:
```
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

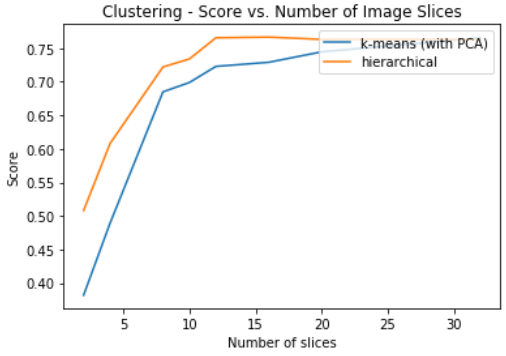
In [71]:
```
import sys

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 [72]:
```
slice_opt_results_3clusters = optimize_slices(datafolder, feature_funcs, n_clusters=3, list_n_slices=[2,4,8,10,12,16,20,32])
```

```
Running slice optimization for 3 clusters:
Done.%
```

In [73]:
```
slice_opt_results_4clusters = optimize_slices(datafolder, feature_funcs, n_clusters=4, list_n_slices=[2,4,8,10,12,16,20,32])
```

```
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 [116]:
```
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 [131]:
```
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 one time ran it up to 64 slices, but that take forever. k-means flattens and hierarchical goes drops drastically)

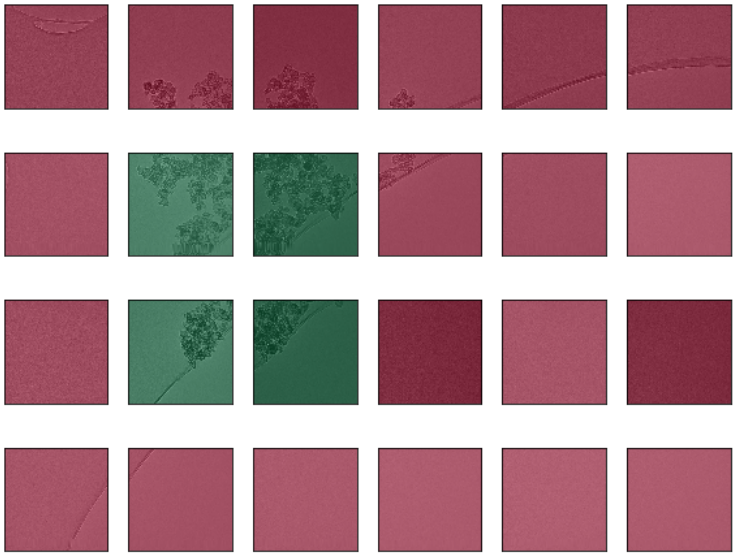
10B. Look at some heatmaps for 2 clusters

In [140]:
```
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, subti
tle=extratitle)
```

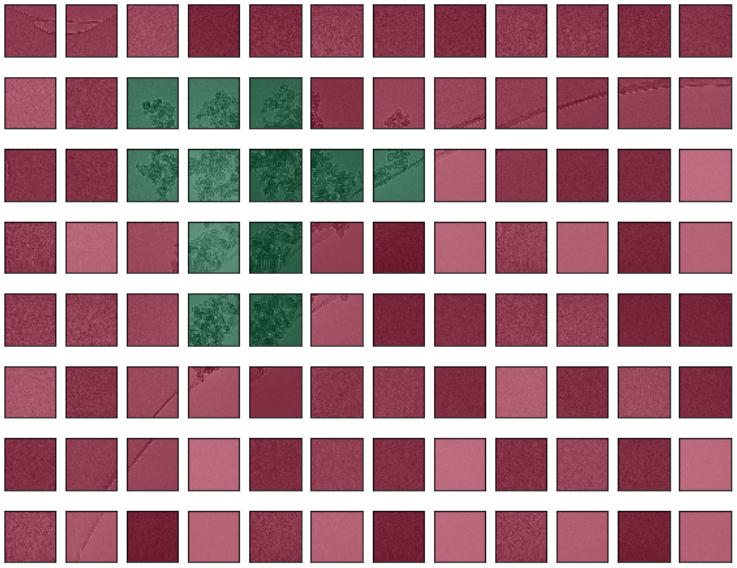
In [134]:
```
def show_multiple_from_slicegridsearch(results, list_num_clust, list_num_slices, n_img_rows, n_img_cols,
                                       heat_col='kmeans', fig_size=(12,10)):
    for n_slices in list_num_slices:
        for n_clust in list_num_clust:
            show_from_slicegridsearch(results, n_clust, n_slices, n_img_rows, n_img_cols, heat_col,  fig_size=fig_size)
```

In [132]:
```
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_co
l='kmeans',  fig_size=(10,8))
```

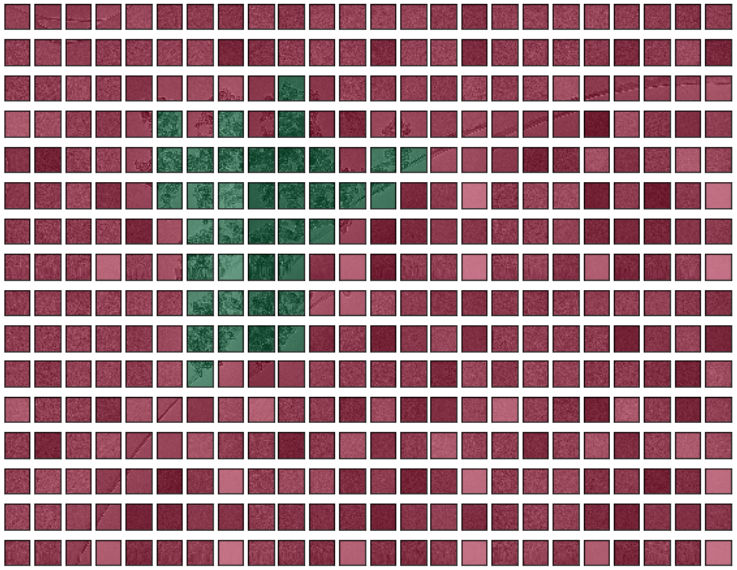
Heats from: kmeans - 2x2 slices, 2 clusters



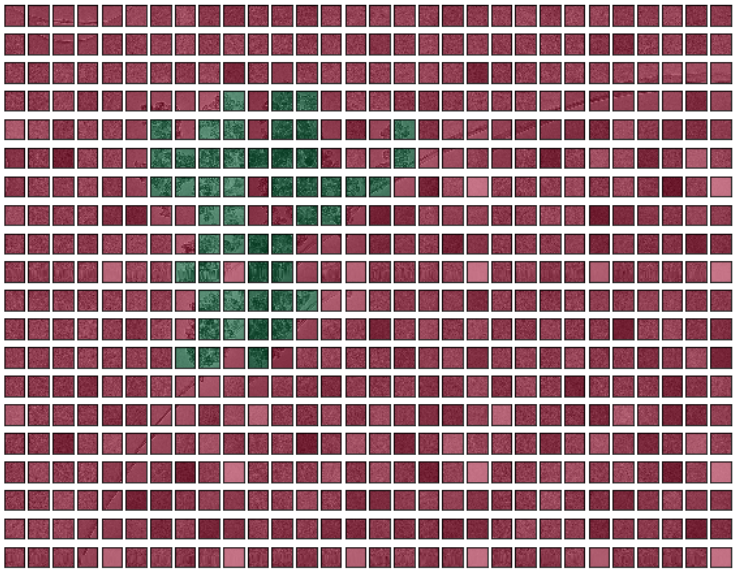
Heats from: kmeans - 4x4 slices, 2 clusters



Heats from: kmeans - 8x8 slices, 2 clusters

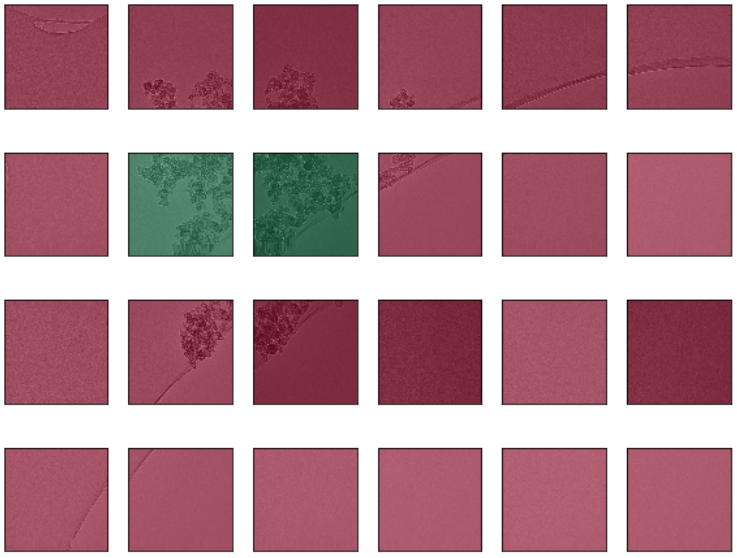


Heats from: kmeans - 10x10 slices, 2 clusters

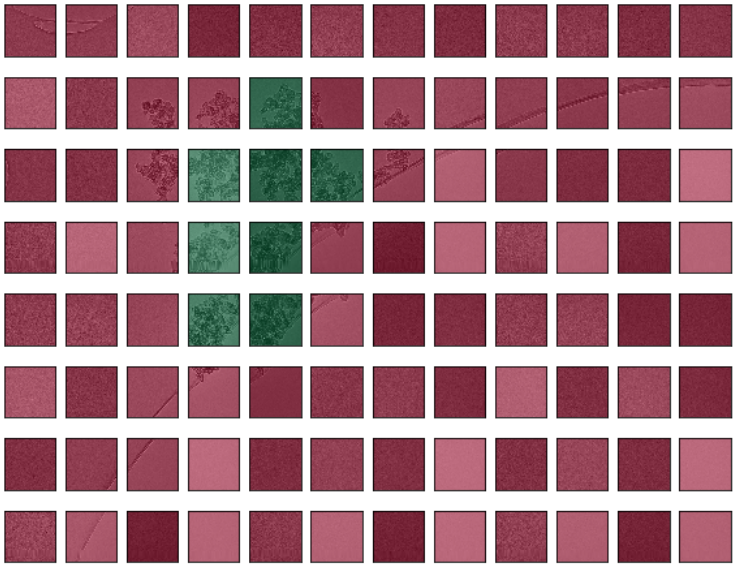


In [135]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [2], [2,4,8, 10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_col=
'hierarchical',  fig_size=(10,8))
```

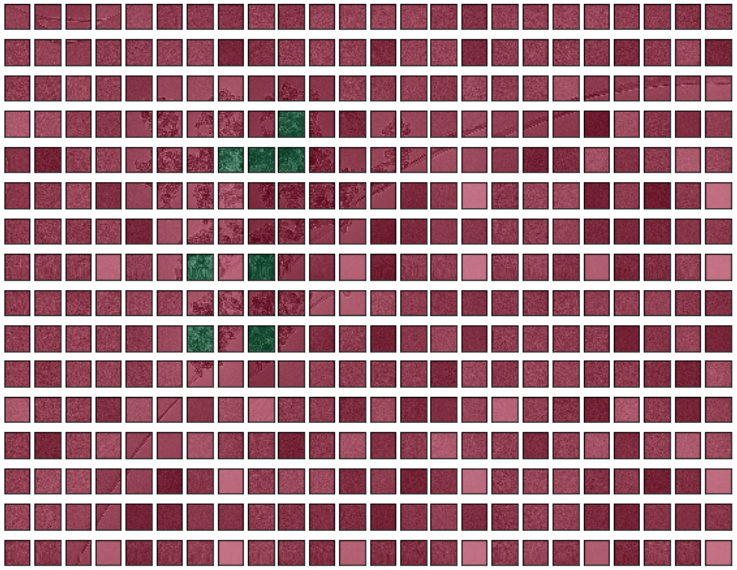
Heats from: hierarchical - 2x2 slices, 2 clusters

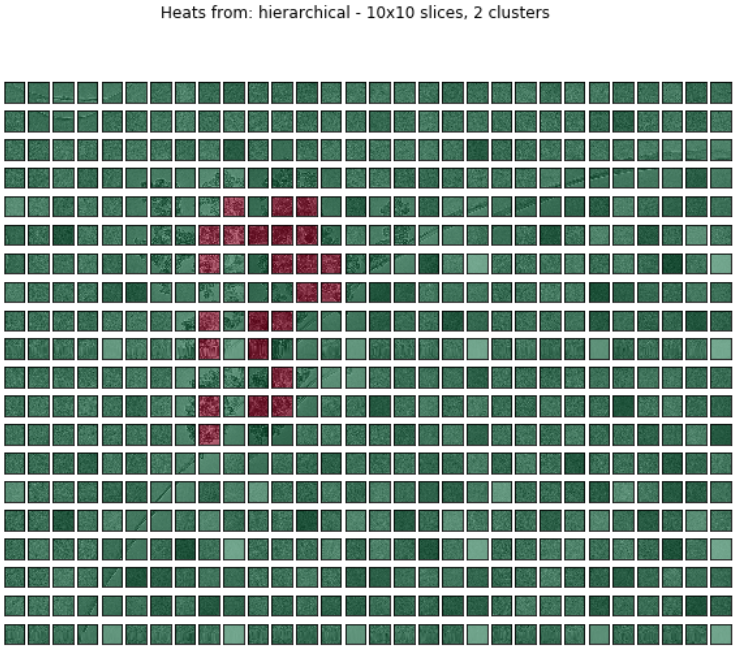


Heats from: hierarchical - 4x4 slices, 2 clusters



Heats from: hierarchical - 8x8 slices, 2 clusters

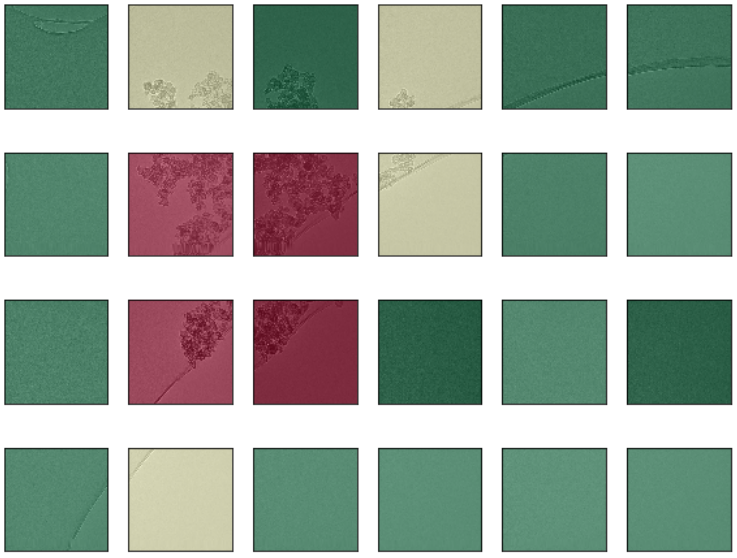




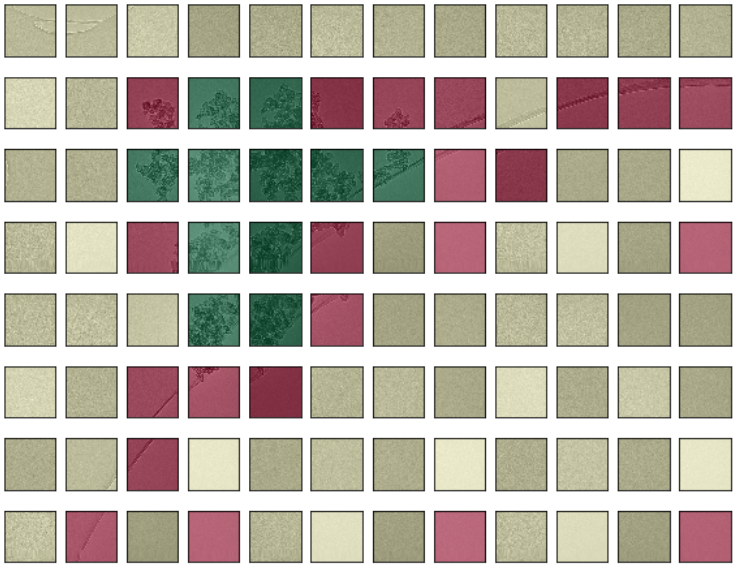
10c. Look at some heatmaps for3 clusters

In [138]:
```
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_co
l='kmeans',  fig_size=(10,8))
```

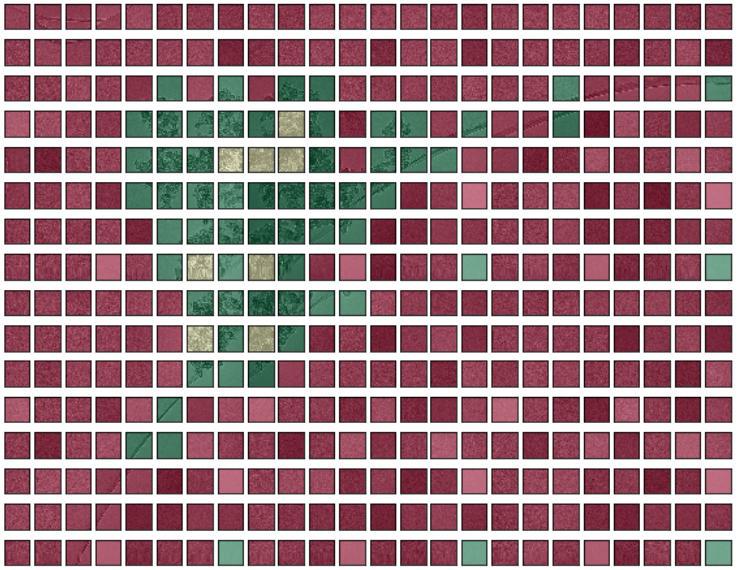
Heats from: kmeans - 2x2 slices, 3 clusters



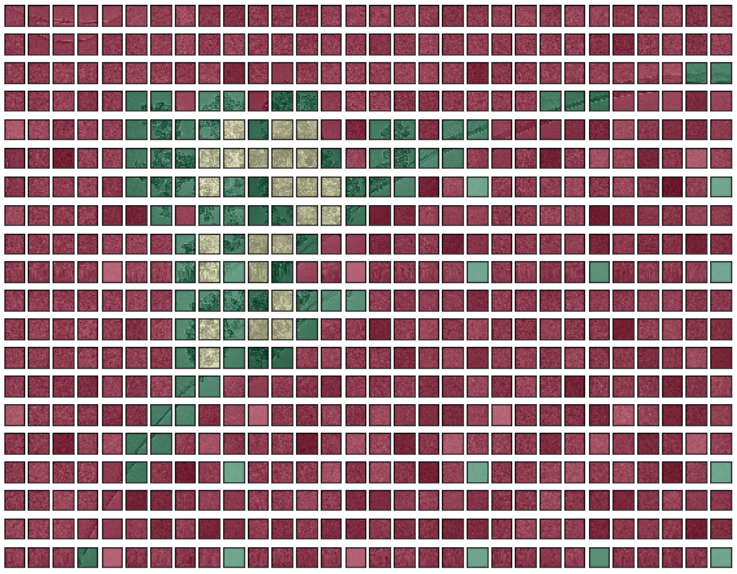
Heats from: kmeans - 4x4 slices, 3 clusters



Heats from: kmeans - 8x8 slices, 3 clusters

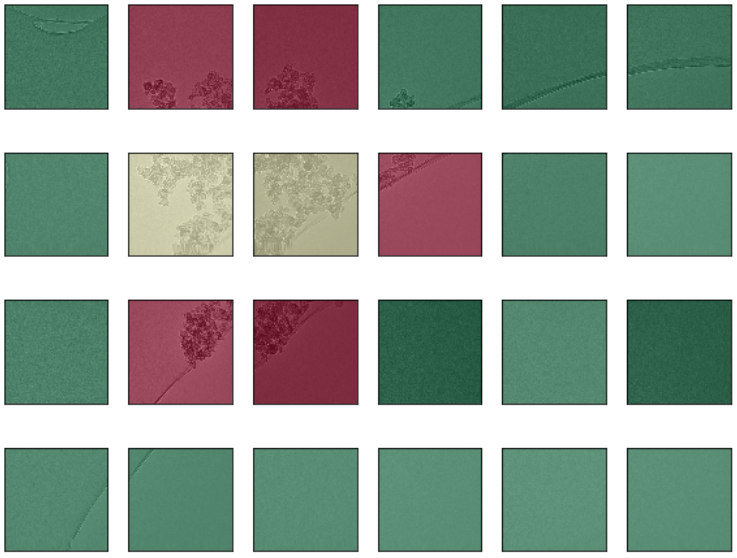


Heats from: kmeans - 10x10 slices, 3 clusters

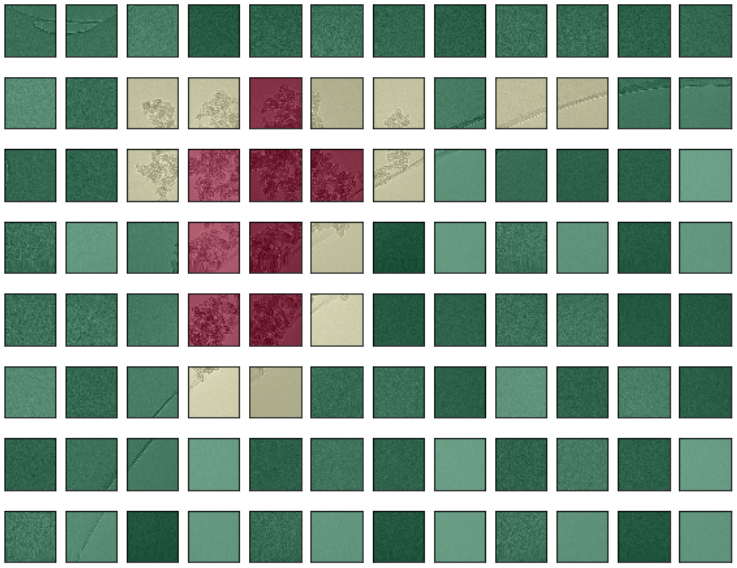


In [139]:
```
show_multiple_from_slicegridsearch(slice_grid_search, [3], [2, 4, 8 ,10], n_img_rows=n_tiles_y, n_img_cols=n_tiles_x, heat_co
l='hierarchical',  fig_size=(10,8))
```

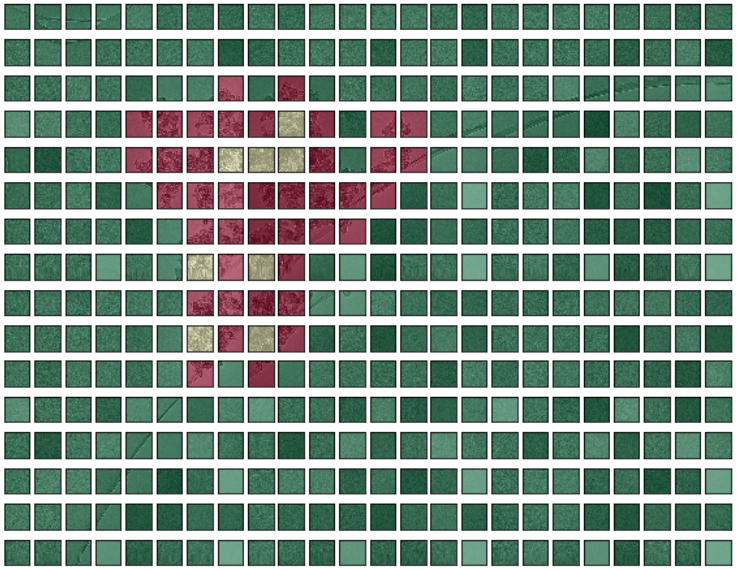
Heats from: hierarchical - 2x2 slices, 3 clusters



Heats from: hierarchical - 4x4 slices, 3 clusters



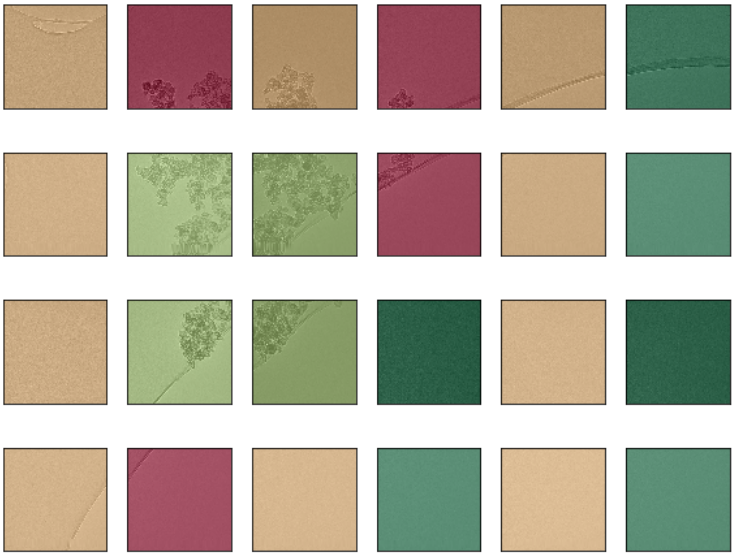
Heats from: hierarchical - 8x8 slices, 3 clusters



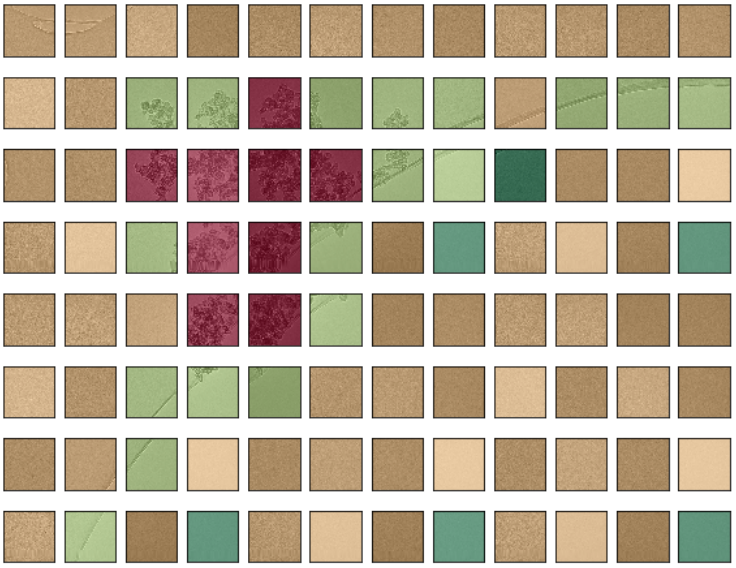
10D. Look at some heatmaps for 4 clusters

In [136]:
```
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_co
l='kmeans',  fig_size=(10,8))
```

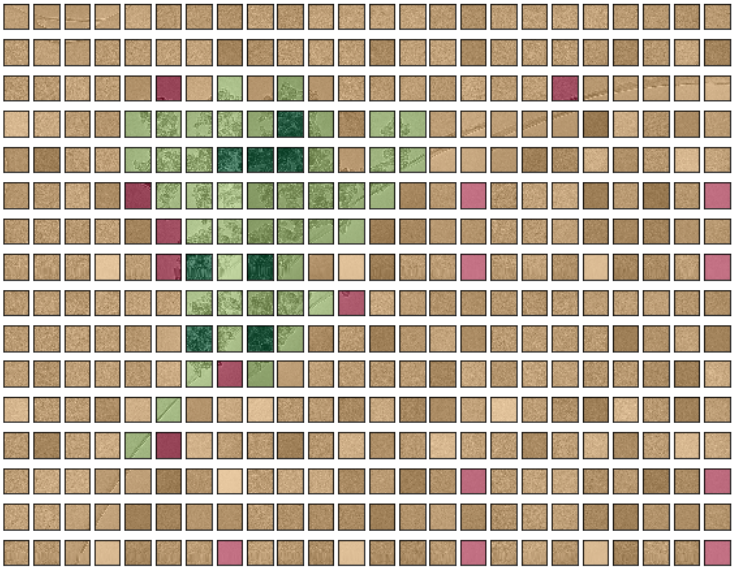
Heats from: kmeans - 2x2 slices, 4 clusters



Heats from: kmeans - 4x4 slices, 4 clusters



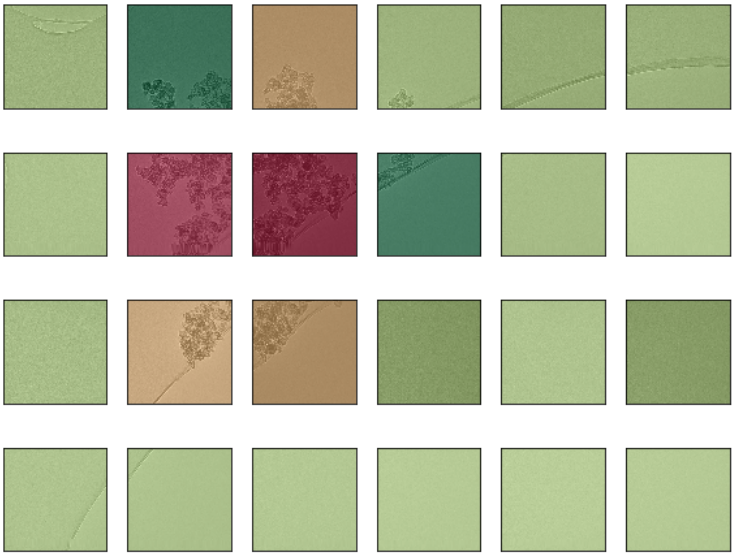
Heats from: kmeans - 8x8 slices, 4 clusters



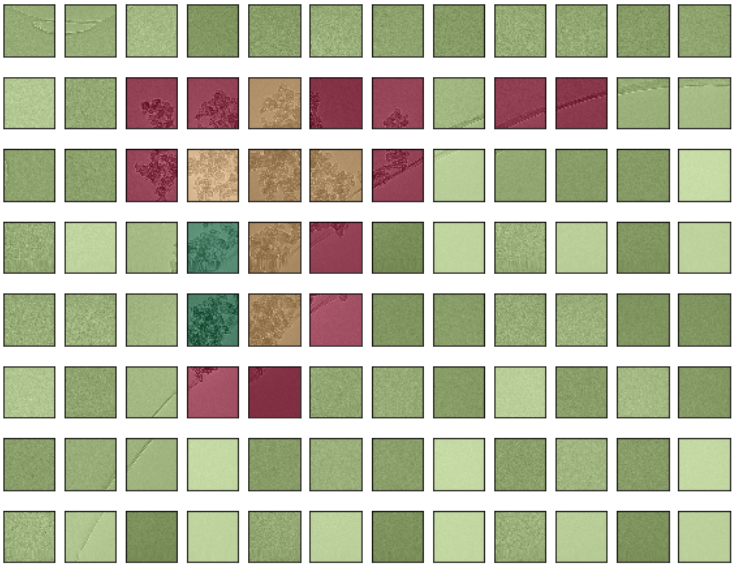
Heats from: kmeans - 10x10 slices, 4 clusters

In [137]:
```
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_co
l='hierarchical',  fig_size=(10,8))
```

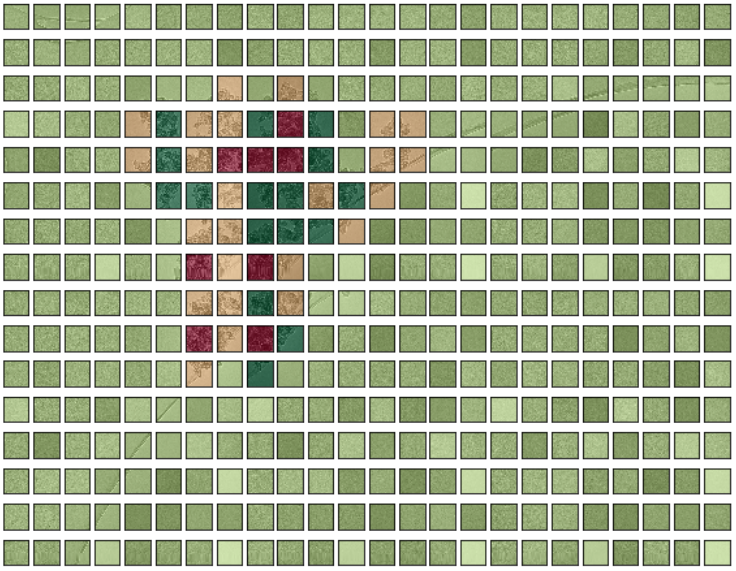
Heats from: hierarchical - 2x2 slices, 4 clusters

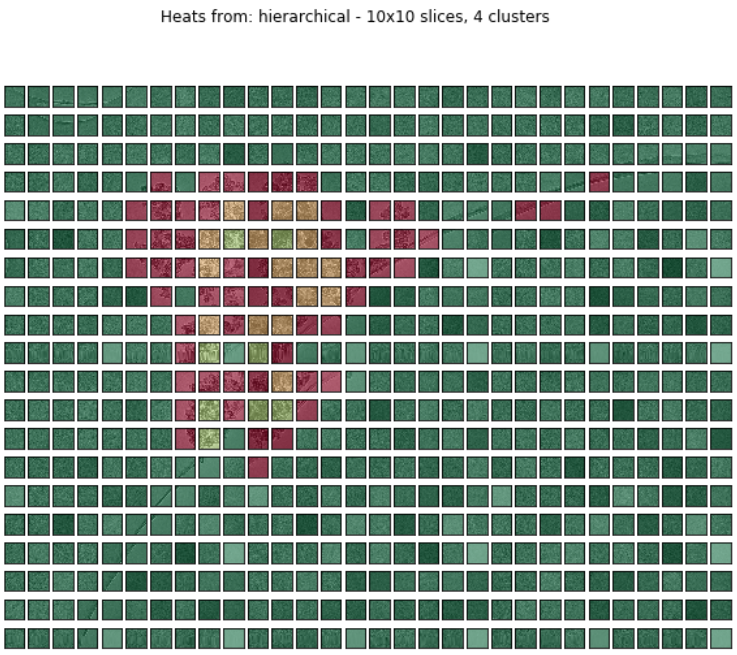


Heats from: hierarchical - 4x4 slices, 4 clusters



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 !)