Clustering with satellite images

March 27, 2018

1 Applying clustering to identify land use in Satellite imagery

In [55]:
```
import os

#data manipulation
import numpy as np
import pandas as pd

#reading and displying images
import matplotlib.pyplot as plt
import matplotlib.patches as pat
import seaborn as sns

#displying data
from IPython.display import display

#the K-means implementation
from sklearn.cluster import KMeans

#guassaian smoothing
from scipy.ndimage import gaussian_filter

#inline plots
%matplotlib inline
plt.rcParams["figure.figsize"] = (20,20)
```

2 The Data

Lets try some satellite images from: https://apps.sentinel-hub.com/sentinel-playground

We load several images to try. One is the natural image; what you see with your eyes. The others include several different spectra highlighing argicultural, urban and vegetation.

In [24]:
```
files = ["Sentinel-2 image on 2018-012-natural.jpg",
         "Sentinel-2 image on 2018-01-12-agric.jpg",
         "Sentinel-2 image on 2018-01-12-urban.jpg",
         "Sentinel-2 image on 2018-01-12-vegetation.jpg"
```

```
                ]

        names = ["Natural",
                 "Agricultural",
                 "Urban",
                 "Vegetation"]
```

Let's read the files into a dict

```
In [25]: file_dir = "../data/ghana_data/"

         images = [plt.imread(file_dir + file) for file in files]
         images = dict(zip(names, images))
```

2.1 Pre-processing the image

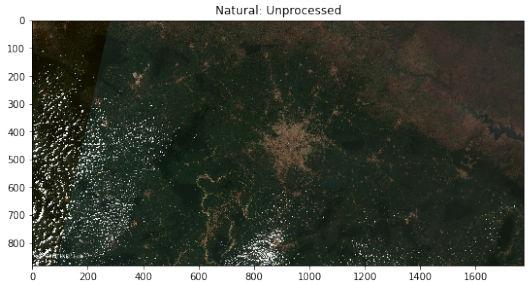
We can try to pre-process the image by applying a gaussian smoothing function. This will mean we lose some fine detail but we are not interested in that anyway.

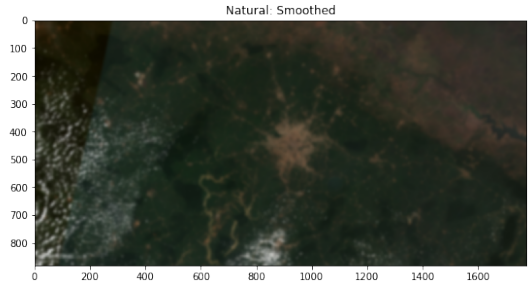
This will give us some idea if pre-processing is best for the images.

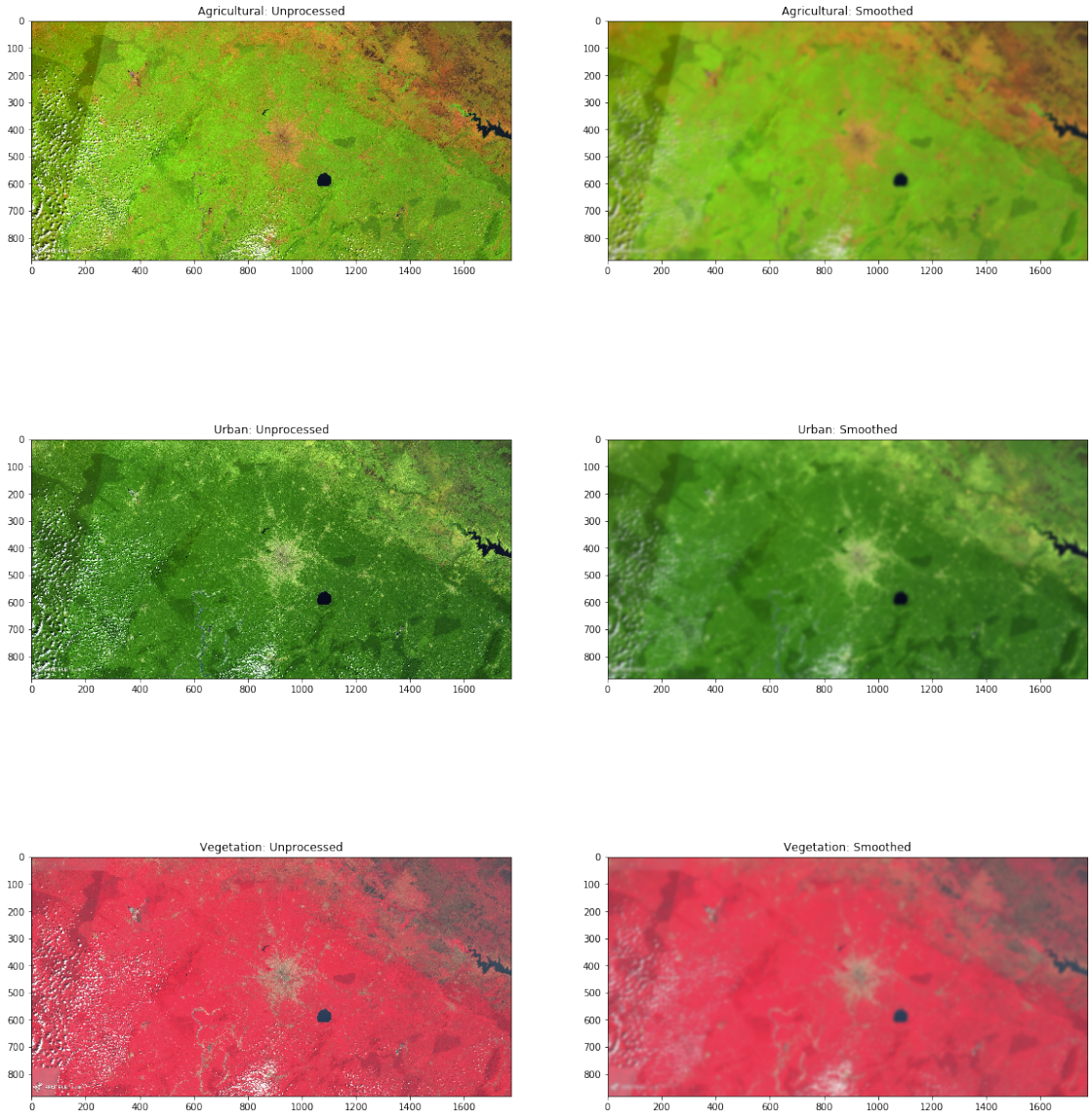
```
In [26]: smooth_imgs = []

         for name in names:
             smooth_imgs.append(gaussian_filter(images[name], sigma = [5,5,0]))

         smooth_images = dict(zip(names, smooth_imgs))
```

Ok, let's look at the images

```
In [27]: for name in names:

             fig, axs = plt.subplots(1,2)
             axs[0].imshow(images[name])
             axs[0].set_title(name + ": Unprocessed")
             axs[1].imshow(smooth_images[name])
             axs[1].set_title(name + ": Smoothed")
             plt.show()
```







We can see the main city in the centre of the images with a mountainous area in the top and right of the image. The white speckled bits are cloud. Overall it is a pretty clear image, there is a darker region to the left which is where one satellite image has been stitched together with another.

We can see the difference in fine detail lost after the smoothing. But that might suit us as we are not interested in the fine detail

2.2 Clustering

Here we define a function that runs our k-means clustering algorithm.

Clustering is an unsupervised machine learning approach. This means the data does not contain labels, so we do not tell the algorithm what class a particular observation (in this case a pixel in the image) should have. Instead, the algorithm looks at the distribution of the various features

(in our case the amount of red, green and blue of each pixel) and tells us how the data is best grouped into classes.

Clustering algorithms can be sensitive to the starting parameters, so we should try the approach with a few different parameters.

K-Means is a pretty straightfoward approach to clustering.

In K-Means, k is the number of clusters we want it to find, which we define beforehand. As K-Means is sensitive to starting parameters, we will try several different values for K.

The broad steps for the algorithm are as follows

1. Select the centroids for each of the K clusters - this can be done by randomly selecting an observation in our dataset or by defining them beforehand. Here we select them randomly.
2. For each observation, it calculates the euclidian distance in the feature space to the centroid of each cluster each datapoint is assigned to the cluster that has shortest euclidian distance. Simply put: it assigns each datapoint to the closest cluster.
3. The centroids for each cluster are recalculated based upon the mean (hence the name) of the features across all the observations' grouped in that cluster.
4. Repeat from step 2 until a stopping condition is met. Examples of these are: no observation changes cluster, the sum of the euclian distances of each observation and the centroid of its cluster drops below a threshold or a maximum number of iterations is reached.

Below is the code to compute clusters. This uses the scikit-learn implementation of K-Means documented here: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

2.3 Functions

2.3.1 Creating clusters

Here we create two functions. The first, cluster_image contains the code to run the k-means algorithm on the data. The second runs the k_means with multiple values for k and gets the results

```
In [28]: def cluster_image(groups, img, method = "random"):
             """cluster_image
             Takes an image, represented as a numpy array and attempts to cluster the pixels
             in the image into a specified number of groups.
             By default uses random starting clusters with a specified random seed

             Args:
                 groups (int): The number of groups to cluster the data into (k)
                 img (Array): The image of dimensions (x,y,z) where z is the features to clust
                 method (String): Initial starting method to use, random by default.
                     See: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMe

             Returns:
                 cluster_labels (Array): Contains cluster labels for img in a 2-D array of the
                     dimensions of img

             """
```

```
            #put into the right shape
            dims = np.shape(img)
            img_matrix = np.reshape(img, (dims[0] * dims[1], dims[2]))

            #cluster
            cl = KMeans(n_clusters = groups, init = method)
            img_groups = cl.fit_predict(img_matrix)

            #create image
            cluster_groups = np.reshape(img_groups, (dims[0], dims[1]))

            return cluster_groups
```

```
In [29]: def cluster_ks(image, ks):

            """cluster_ks
            Wrapper for cluster image. Repeats clustering for a range of values.

            Args:
                image (Array): The image of dimensions (x,y,z) where z is the features to clu
                ks (iterable): integer values of k to cluster with

            Returns:
                (dict): key:value pair where key is k clusters and value is the results in a

            """

            cluster_labels = []

            for k in ks:

                #get cluster groups
                group_labels = cluster_image(groups = k, img = image)
                cluster_labels.append(group_labels)


            clusters = [str(k) for k in ks]
            return dict(zip(clusters,cluster_labels))
```

2.3.2 Visualising results

plt_results plots the group of each pixel, formatted in an array of the same size and shape of the original image and organised them by number of clusters for the smoothed and unsmoothed image. It also plots the pixel counts, ratios and percentages of each group, from the output of calc_counts.

```
In [181]: def plt_results(ks,
                          imgs,
```

```
            smoothed_imgs,
            img_name,
            file_type = ".png",
            save_dir = "../results/clustering/"
           ):

    """plt_results

    Plot results from smoothed and unsmoothed images side by side

    Args:
        ks (iterable): the value for k used to cluster
        img (dict): cluster results from unsmoothed image
        smoothed_img (dict): cluster results from smoothed image
        img_name (string): name of the image the results are for
        file_type (string): image file extention for saving, must be something that
        save_dir (string): directory to save the images to

    Returns:
        figs (List): the figures created from the results
    """

    figs =[]
    for k in range(3,10):
        fig, axs = plt.subplots(1,2)

        im = axs[0].imshow(imgs[str(k)])
        handle = make_legend_handles(img = im,
                                     n_clusters = k
                                    )
        axs[0].legend(handles = handle)
        axs[0].set_title(img_name + ": {} clusters".format(k))

        im = axs[1].imshow(smoothed_imgs[str(k)])
        handle = make_legend_handles(img = im,
                             n_clusters = k
                            )
        axs[1].legend(handles = handle)
        axs[1].set_title(img_name + ", smoothed: {} clusters".format(k))


        plt.show()

        #get the counts
        img_res_df = calc_counts(imgs[str(k)])
        smooth_img_res_df = calc_counts(smoothed_imgs[str(k)])
        #put them together
        res_df = pd.concat([img_res_df, smooth_img_res_df],
```

```
                                    axis = 1,
                                    keys = ["unsmoothed", "smoothed"]
                                   )
                    display(res_df)

                    if save_dir is not None:
                        img_file = save_dir + img_name + "_{}-clusters".format(k) + file_type
                        fig.savefig(img_file)
                        res_file = save_dir + img_name + "_{}-clusters.csv".format(k)
                        res_df.to_csv(res_file)
                    figs.append(fig)

                return
```

In [171]:
```
def make_legend_handles(img, n_clusters):
    """make_legend_handles

    creates handles for use with legend

    Args:
        img (ListedColourmap): the image for the legend
        n_clusters (int): number of clusters

    """
    #create colours
    colours = [img.cmap(img.norm(cluster)) for cluster in range(n_clusters)]
    #use a list of Patch objects for the handle
    handles = []
    for cluster in range(n_clusters):
        handles.append(pat.Patch(color = colours[cluster], label = "Cluster {}".forma

    return handles
```

In [31]:
```
def calc_counts(results):
    """calc_counts

    Computes and returns counts and ratios for number of pixels in the image within e

    Args:
        results (Array): 2D array of cluster labels
    Returns
        df (DataFrame): contains the counts, ratios and percentages of each cluster

    """
    uni, counts = np.unique(results, return_counts = True)
    df = pd.DataFrame({"cluster": uni,
                       "pixel_count" : counts})
```

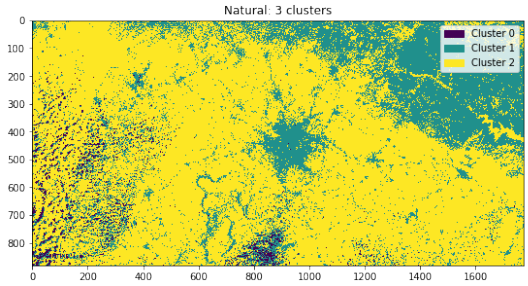
```
    df["ratio"] = df["pixel_count"].divide(df["pixel_count"].sum())
    df["% total"] = df["ratio"].multiply(100)
    df.set_index("cluster", inplace = True)
    return df
```

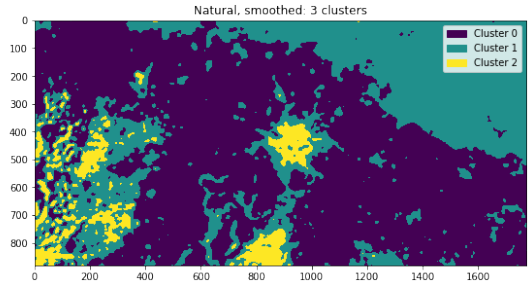
2.4 Natural image

Lets run this with the Natural image

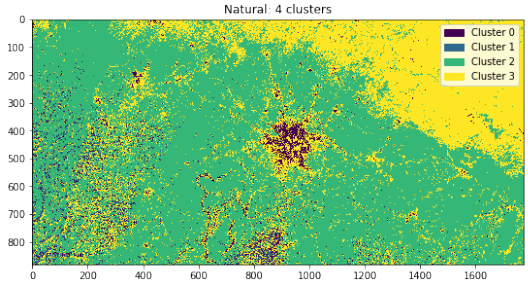
```
In [32]: nat_results = cluster_ks(images["Natural"], range(3,10))
         nat_smooth_results = cluster_ks(smooth_images["Natural"], range(3,10))
```

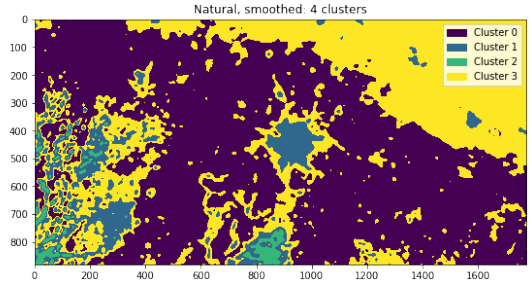
```
In [185]: plt_results(ks = range(3,10),
                      imgs = nat_results,
                      smoothed_imgs = nat_smooth_results,
                      img_name = "Natural"
                     )
```



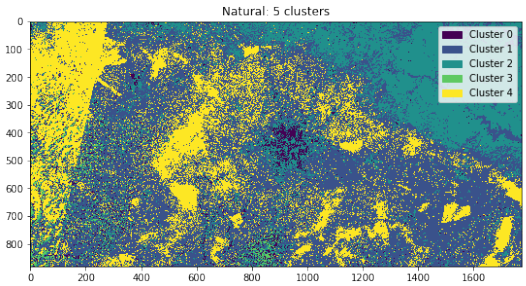


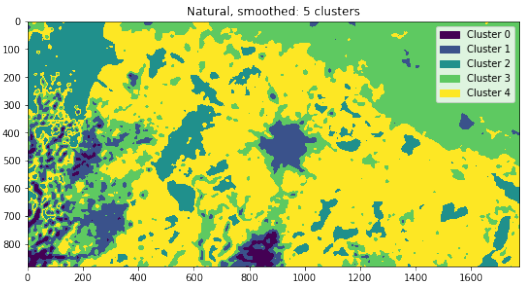
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	31707	0.020253	2.025295	923487	0.589880	58.988023
1	405611	0.259085	25.908531	561515	0.358669	35.866948
2	1128232	0.720662	72.066175	80548	0.051450	5.145029



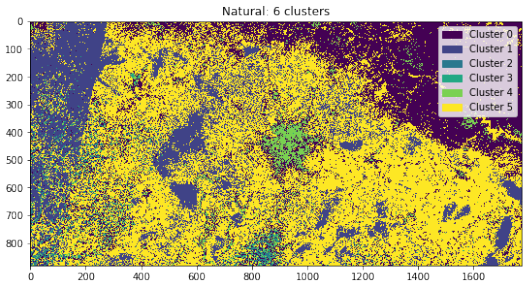


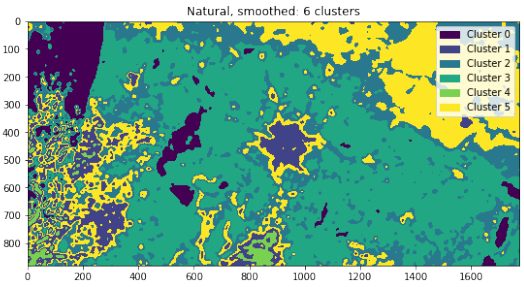
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	54111	0.034564	3.456357	888293	0.567400	56.739996
1	25227	0.016114	1.611383	111381	0.071145	7.114497
2	1008395	0.644115	64.411549	20337	0.012990	1.299032
3	477817	0.305207	30.520712	545539	0.348465	34.846476





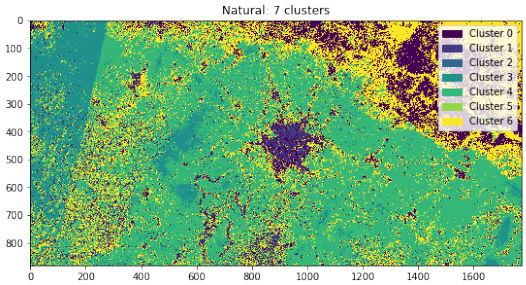
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	48930	0.031254	3.125419	19790	0.012641	1.264092
1	735198	0.469610	46.961004	106474	0.068011	6.801060
2	386250	0.246718	24.671841	185715	0.118626	11.862604
3	24851	0.015874	1.587365	496749	0.317300	31.729999
4	370321	0.236544	23.654371	756822	0.483422	48.342244

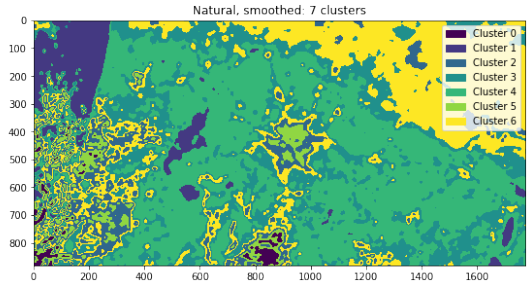




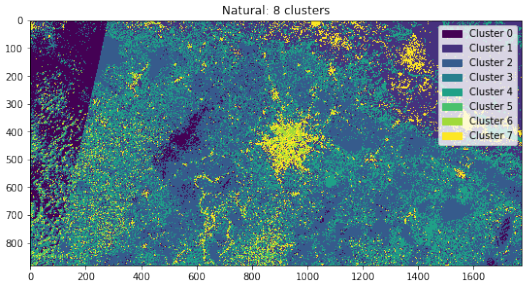
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	401526	0.256476	25.647600	101382	0.064758	6.475807
1	313487	0.200241	20.024081	92097	0.058827	5.882725
2	20566	0.013137	1.313660	363284	0.232049	23.204880

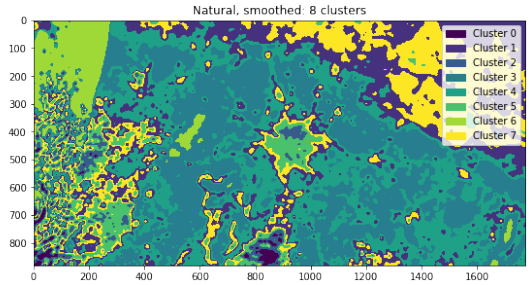
3	17406	0.011118	1.111814	654559	0.418102	41.810163
4	65889	0.042087	4.208681	18071	0.011543	1.154291
5	746676	0.476942	47.694165	336157	0.214721	21.472134





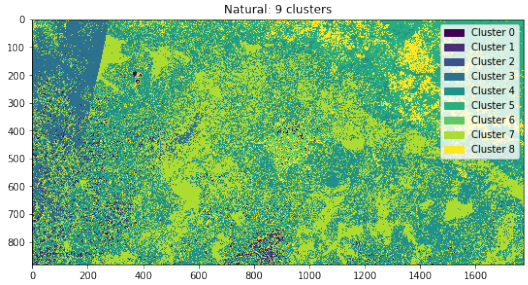
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	207984	0.132850	13.285044	11942	0.007628	0.762799
1	49415	0.031564	3.156399	91254	0.058289	5.828878
2	15533	0.009922	0.992175	101330	0.064725	6.472486
3	161831	0.103370	10.337006	360554	0.230305	23.030500
4	737607	0.471149	47.114880	588147	0.375681	37.568075
5	20042	0.012802	1.280189	46845	0.029922	2.992239
6	373138	0.238343	23.834307	365478	0.233450	23.345023

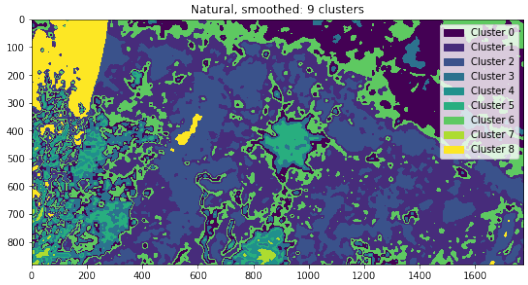




	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	118691	0.075814	7.581425	8668	0.005537	0.553671
1	322030	0.205698	20.569768	294398	0.188048	18.804765
2	554039	0.353894	35.389416	32278	0.020618	2.061767
3	13372	0.008541	0.854141	399015	0.254872	25.487209
4	420302	0.268469	26.846923	418435	0.267277	26.727668

5	19406	0.012396	1.239564	86371	0.055170	5.516975
6	32566	0.020802	2.080164	75070	0.047951	4.795120
7	85144	0.054386	5.438600	251315	0.160528	16.052825





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	16246	0.010377	1.037718	292586	0.186890	18.689023
1	9929	0.006342	0.634218	457351	0.292134	29.213439
2	18302	0.011690	1.169046	303629	0.193944	19.394398
3	103782	0.066291	6.629108	94648	0.060457	6.045671
4	480642	0.307012	30.701159	17643	0.011270	1.126952
5	314930	0.201163	20.116253	52491	0.033529	3.352879
6	45112	0.028815	2.881543	272384	0.173986	17.398614
7	440536	0.281394	28.139376	5181	0.003309	0.330938
8	136071	0.086916	8.691578	69637	0.044481	4.448085

2.4.1 Results

The most stark difference is how much noiser our unsmoothed images look. If we wanted to perform some kind of edge detection or segmentation, we could do that on our smoothed images but not the unsmoothed.

As for picking out the main city. It looks like using three clusters works ok, as do four and five. However, there are still things like clouds and mountains that it is being grouped with,so not perfect.

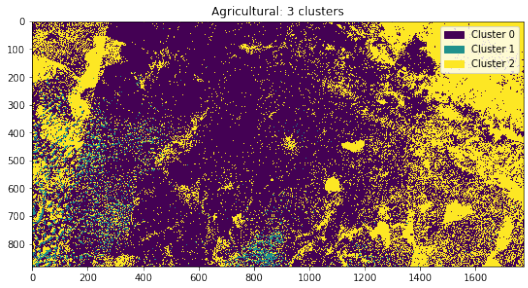
Six clusters and upwards appear to be less useful as multiple clusters cover the city - not ideal for gross segmentation of land use.

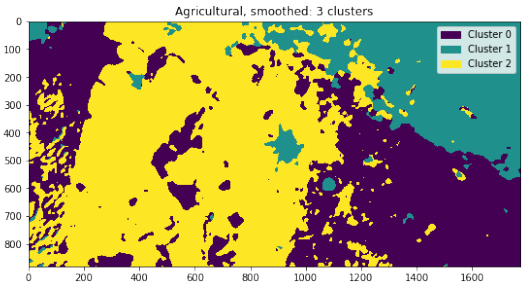
3 Agricultural

Lets repeat the steps above with the agricultural image. This image has already been pre-processed to identify some land use which might make our job easier

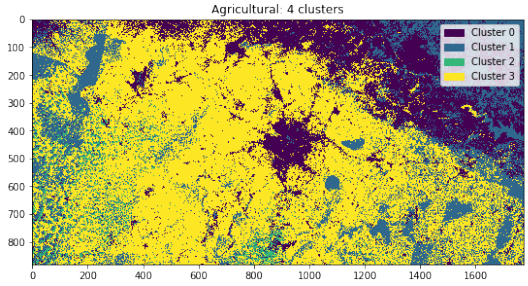
```
In [40]: agric_results = cluster_ks(images["Agricultural"], range(3,10))
         smooth_agric_results = cluster_ks(smooth_images["Agricultural"], range(3,10))
```

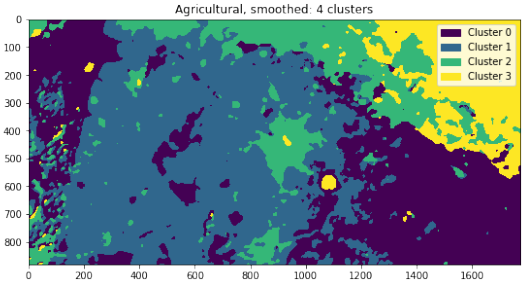
```
In [183]: plt_results(ks = range(3,10),
                      imgs = agric_results,
                      smoothed_imgs = smooth_agric_results,
                      img_name = "Agricultural"
                     )
```



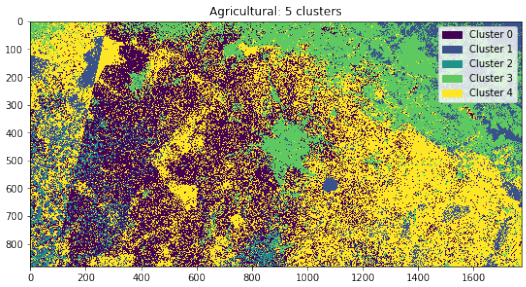


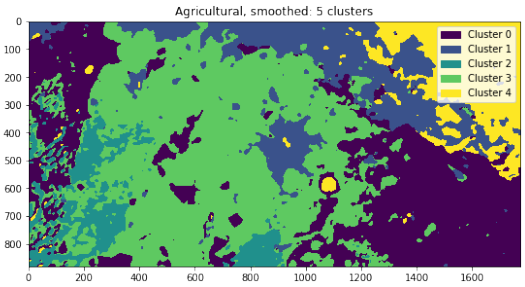
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	1035828	0.661638	66.163840	534331	0.341306	34.130561
1	42475	0.027131	2.713104	306030	0.195478	19.547763
2	487247	0.311231	31.123056	725189	0.463217	46.321676



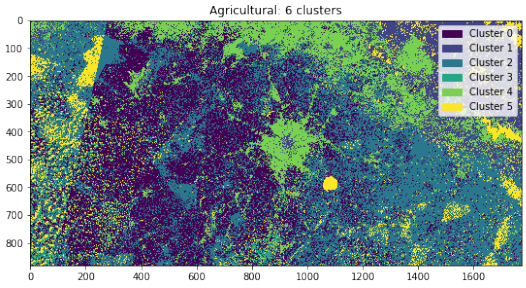


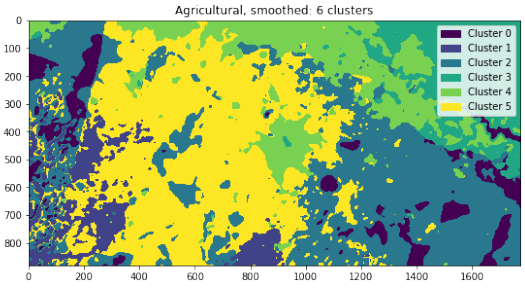
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	325798	0.208105	20.810450	473388	0.302378	30.237808
1	341330	0.218026	21.802561	656061	0.419061	41.906103
2	42798	0.027337	2.733736	250120	0.159765	15.976494
3	855624	0.546533	54.653253	185981	0.118796	11.879595



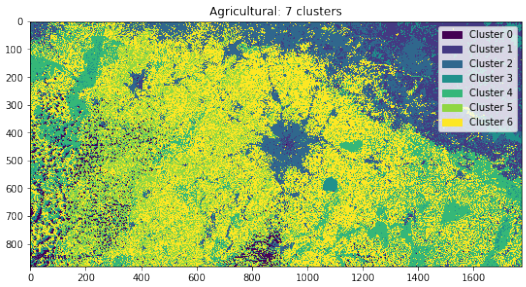


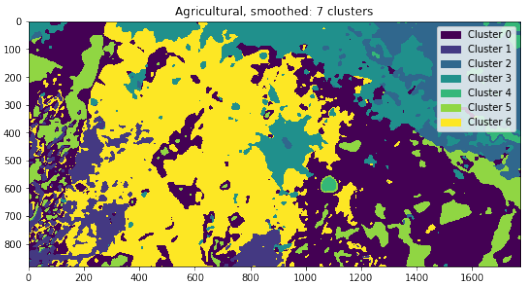
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	466911	0.298241	29.824087	430868	0.275218	27.521829
1	170995	0.109224	10.922360	235208	0.150240	15.023985
2	39292	0.025098	2.509789	102385	0.065399	6.539874
3	317102	0.202550	20.254990	627089	0.400555	40.055508
4	571250	0.364888	36.488774	170000	0.108588	10.858804



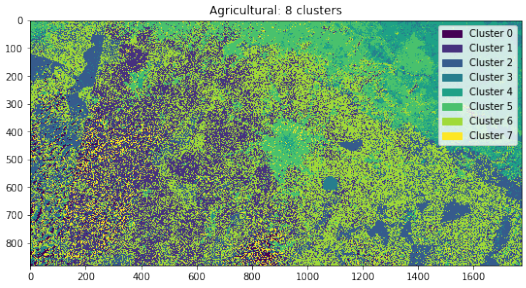


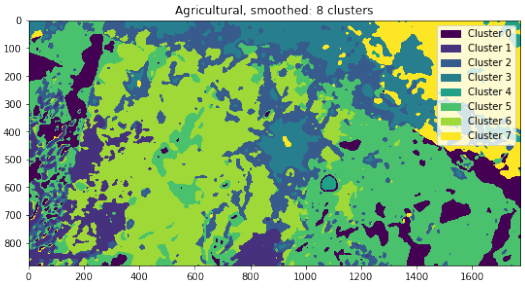
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	456423	0.291542	29.154163	84947	0.054260	5.426016
1	172415	0.110131	11.013063	95932	0.061277	6.127687
2	562272	0.359153	35.915301	462798	0.295614	29.561368
3	39177	0.025024	2.502443	175941	0.112383	11.238287
4	225259	0.143885	14.388490	221468	0.141463	14.146338
5	110004	0.070265	7.026540	524464	0.335003	33.500303



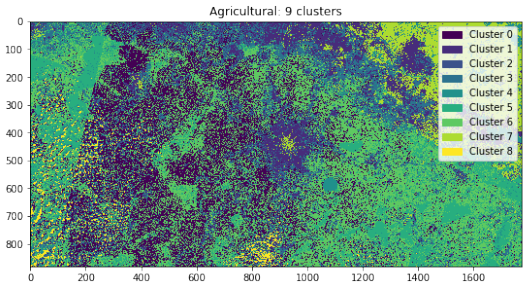


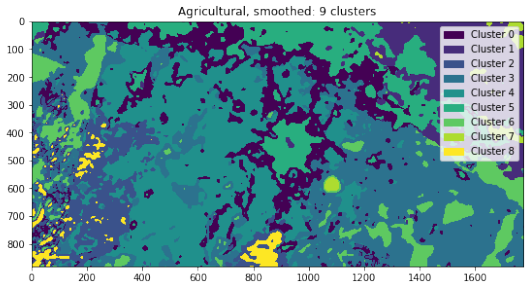
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	36940	0.023596	2.359554	453125	0.289435	28.943502
1	173056	0.110540	11.054007	94661	0.060465	6.046501
2	212125	0.135496	13.549551	174791	0.111648	11.164830
3	66360	0.042388	4.238766	214920	0.137281	13.728083
4	281514	0.179818	17.981796	14254	0.009105	0.910479
5	255889	0.163450	16.344991	115361	0.073687	7.368720
6	539666	0.344713	34.471336	498438	0.318379	31.837884





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	28993	0.018519	1.851937	107923	0.068936	6.893616
1	334557	0.213699	21.369934	91843	0.058665	5.866501
2	219100	0.139951	13.995082	222656	0.142222	14.222222
3	58903	0.037624	3.762448	161134	0.102925	10.292485
4	166210	0.106167	10.616716	13720	0.008764	0.876369
5	216736	0.138441	13.844080	409117	0.261325	26.132477
6	488816	0.312233	31.223276	388558	0.248193	24.819265
7	52235	0.033365	3.336527	170599	0.108971	10.897065





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	312671	0.199720	19.971959	208784	0.133361	13.336144
1	134703	0.086042	8.604197	170444	0.108872	10.887164
2	53278	0.034031	3.403149	114078	0.072868	7.286768
3	199814	0.127632	12.763182	399719	0.255322	25.532177
4	56413	0.036034	3.603398	370773	0.236832	23.683242
5	210703	0.134587	13.458721	157864	0.100836	10.083613
6	413983	0.264433	26.443295	106144	0.067800	6.779981
7	154533	0.098708	9.870844	13699	0.008750	0.875028
8	29452	0.018813	1.881256	24045	0.015359	1.535882

3.0.1 Results

This shows similar results to the natural image. Clustering on the smoothed image more promising.

With three groups, it looks able to seperate out the more mountainous areas with. clouds and agricultural/urban areas. Not quite what we want.

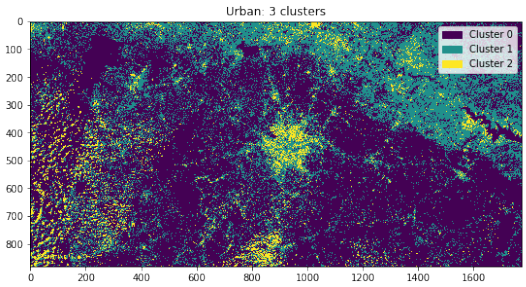
With 4 and 5, you can pick out the urban areas of the city and surrounding towns quite well, but confuses the city with either clouds or the mountains. As you add more clusters it appears to gradually get more noise and appears less useful.

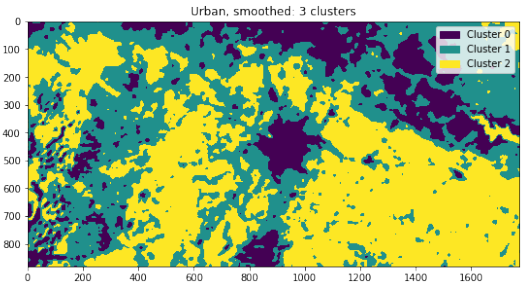
3.1 Urban image

Like the Agricultural image, this has been pre-processed except, to pick out urban areas instead of agricutural.

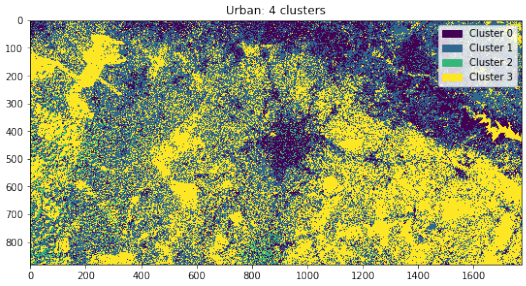
```
In [42]: urban_results = cluster_ks(images["Urban"], range(3,10))
         smooth_urb_results = cluster_ks(smooth_images["Urban"], range(3,10))
```

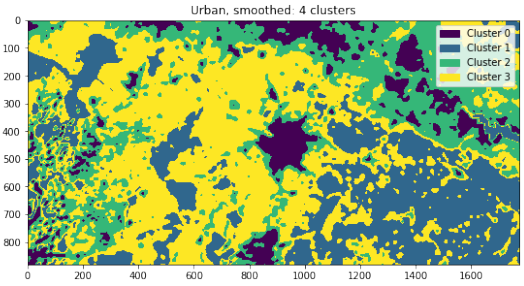
```
In [180]: plt_results(ks = range(3,10),
                      imgs = urban_results,
                      smoothed_imgs = smooth_urb_results,
                      img_name = "Urban"
                     )
```



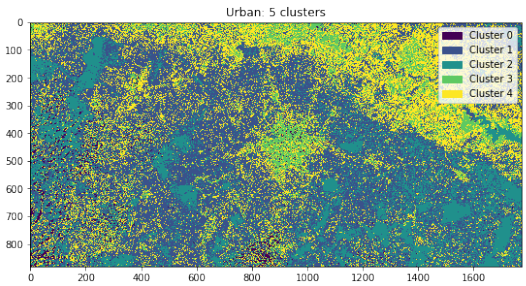


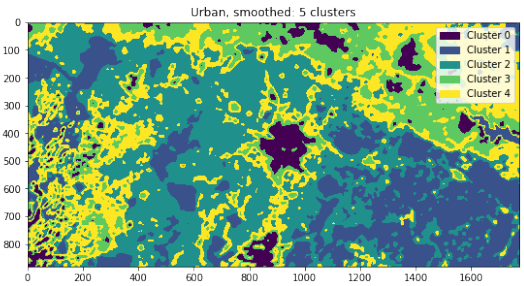
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	996784	0.636699	63.669892	245506	0.156818	15.681773
1	485587	0.310170	31.017023	634039	0.404994	40.499441
2	83179	0.053131	5.313085	686005	0.438188	43.818786



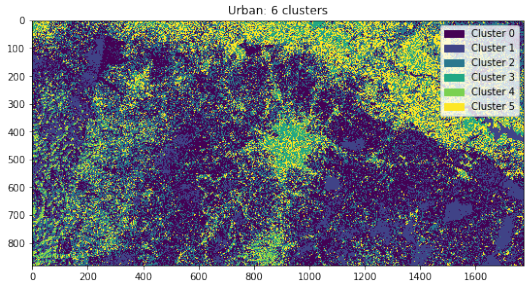


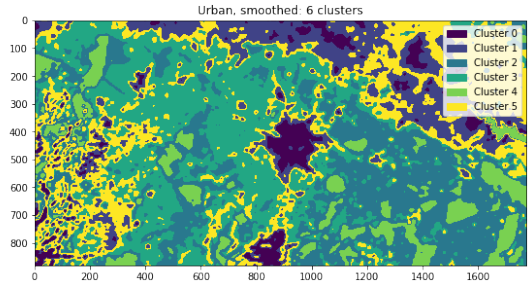
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	245679	0.156928	15.692824	134981	0.086220	8.621954
1	646791	0.413140	41.313979	381441	0.243647	24.364664
2	44123	0.028184	2.818371	396688	0.253386	25.338571
3	628957	0.401748	40.174827	652440	0.416748	41.674811



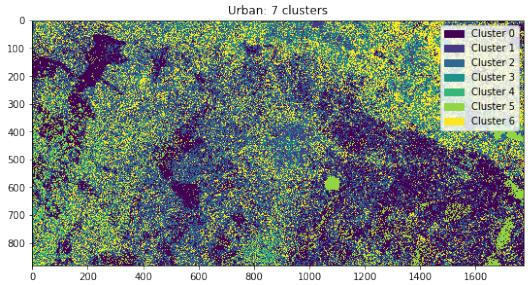


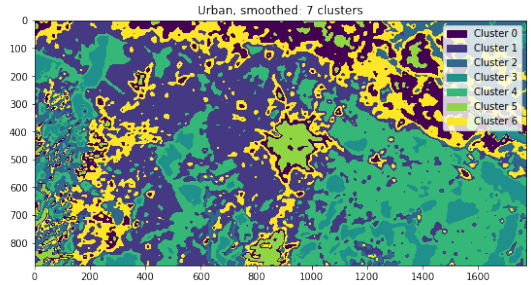
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	33277	0.021256	2.125579	69683	0.044510	4.451024
1	683912	0.436851	43.685095	322151	0.205775	20.577497
2	337135	0.215346	21.534604	560590	0.358079	35.807863
3	144229	0.092127	9.212673	228635	0.146041	14.604133
4	366997	0.234420	23.442049	384491	0.245595	24.559484



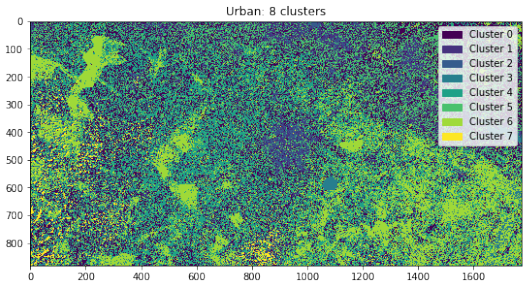


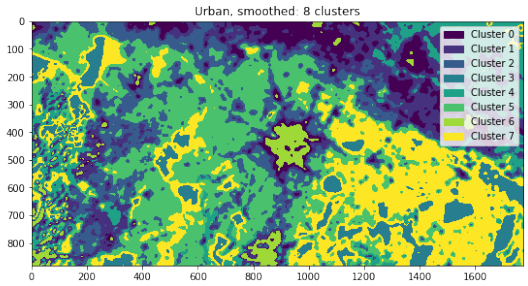
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	586110	0.374380	37.437961	63841	0.040779	4.077864
1	191201	0.122130	12.213024	208443	0.133144	13.314362
2	433135	0.276666	27.666635	385561	0.246278	24.627830
3	94205	0.060174	6.017374	456735	0.291741	29.174092
4	28817	0.018407	1.840695	117581	0.075105	7.510523
5	232082	0.148243	14.824311	333389	0.212953	21.295328



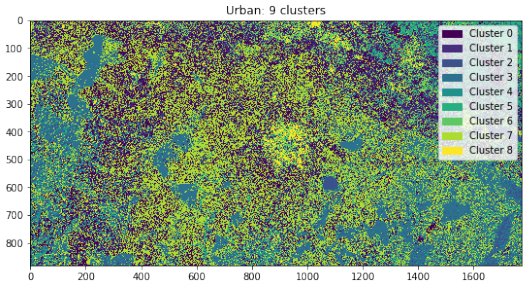


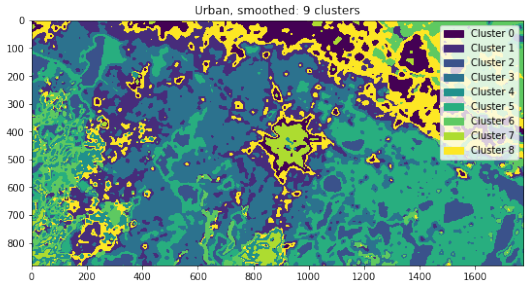
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	414233	0.264593	26.459264	167255	0.106835	10.683466
1	71583	0.045724	4.572387	436062	0.278536	27.853598
2	492879	0.314828	31.482802	155841	0.099544	9.954393
3	171876	0.109786	10.978634	119218	0.076151	7.615087
4	26853	0.017152	1.715244	381672	0.243794	24.379419
5	77075	0.049232	4.923190	52175	0.033327	3.332695
6	311051	0.198685	19.868481	253327	0.161813	16.181342





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	343232	0.219241	21.924052	103357	0.066020	6.601961
1	121583	0.077662	7.766152	213246	0.136212	13.621156
2	51646	0.032989	3.298905	231890	0.148120	14.812047
3	51481	0.032884	3.288365	99176	0.063349	6.334898
4	451754	0.288559	28.855929	156224	0.099789	9.978857
5	216261	0.138137	13.813740	404704	0.258506	25.850596
6	304652	0.194597	19.459743	39195	0.025036	2.503593
7	24941	0.015931	1.593114	317758	0.202969	20.296892





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	280861	0.179401	17.940085	97714	0.062415	6.241513
1	212903	0.135992	13.599246	232663	0.148614	14.861423
2	49311	0.031498	3.149756	96623	0.061718	6.171825
3	287497	0.183640	18.363962	392376	0.250631	25.063141
4	24876	0.015890	1.588962	41365	0.026422	2.642202
5	112454	0.071830	7.183035	311555	0.199007	19.900674
6	119997	0.076648	7.664846	160302	0.102393	10.239341
7	426509	0.272434	27.243397	32135	0.020526	2.052633
8	51142	0.032667	3.266711	200817	0.128272	12.827249

3.1.1 Results

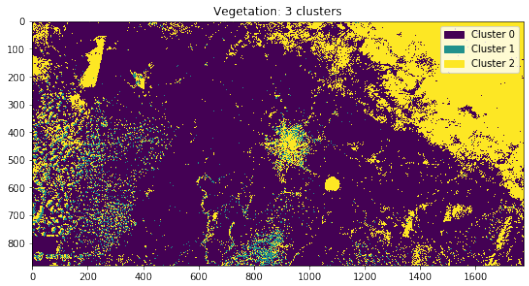
As with the previous two images, four and five clusters show differences between the surrounding grassland and the city in the centre, but struggle to sepearate the city from the clouds while adding more clusters just adds more noise to the result.

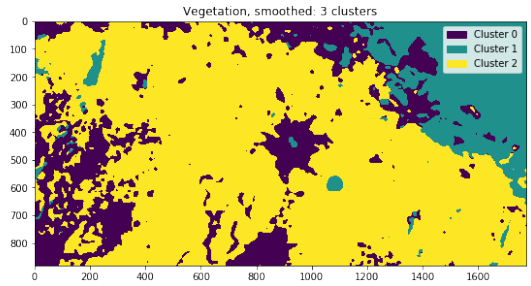
3.2 Vegetation image

This image uses parts of the infared spectrum to show areas of vegetation so may be able to seperate out the urban areas from the city.

```
In [44]: veg_results = cluster_ks(images["Vegetation"], range(3,10))
         smooth_veg_results = cluster_ks(smooth_images["Vegetation"], range(3,10))
```

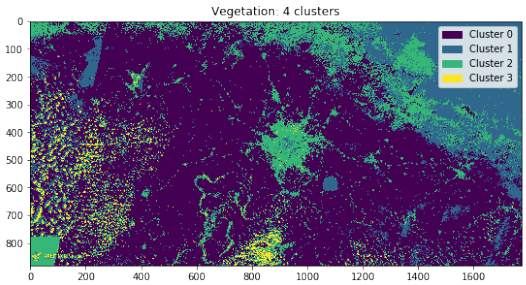
```
In [175]: plt_results(ks = range(3,10),
                      imgs = veg_results,
                      smoothed_imgs = smooth_veg_results,
                      img_name = "Vegetation"
                     )
```

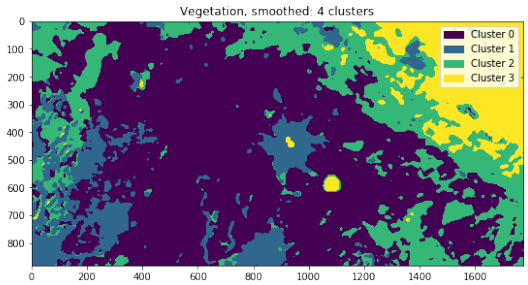




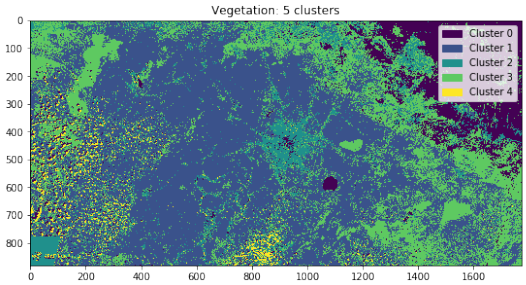
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						

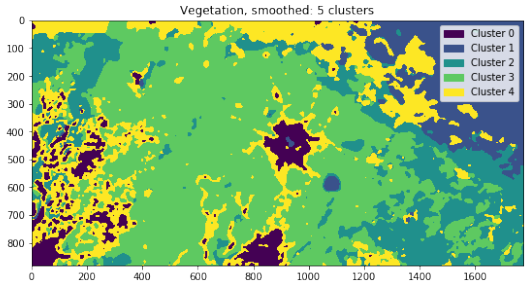
0	1133162	0.723811	72.381080	306839	0.195994	19.599438
1	52897	0.033788	3.378813	234104	0.149535	14.953467
2	379491	0.242401	24.240107	1024607	0.654471	65.447095



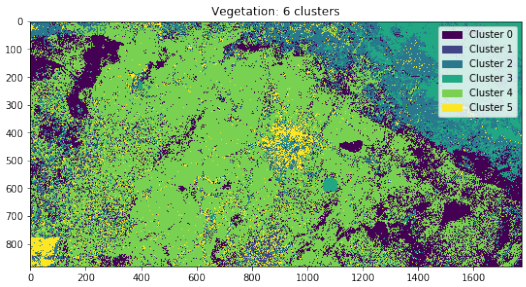


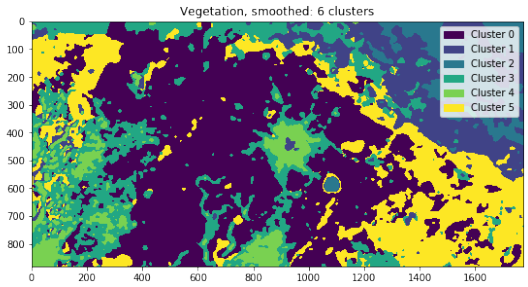
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	1005532	0.642287	64.228674	854565	0.545856	54.585609
1	286481	0.182991	18.299064	184629	0.117932	11.793236
2	238194	0.152147	15.214717	342221	0.218595	21.859474
3	35343	0.022575	2.257545	184135	0.117617	11.761681



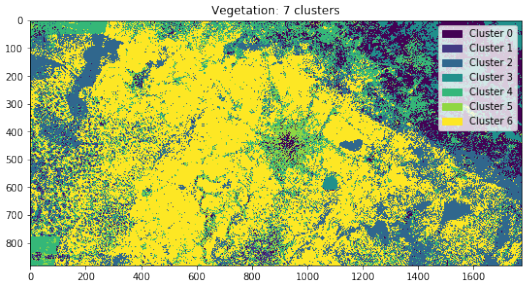


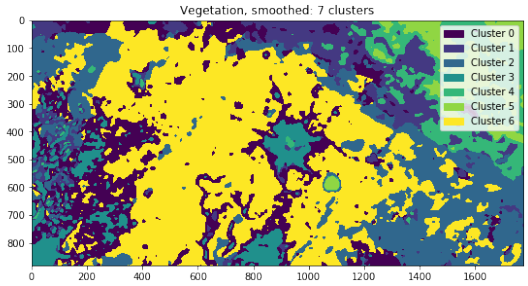
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	201828	0.128918	12.891827	80752	0.051581	5.158059
1	787859	0.503247	50.324742	200379	0.127993	12.799272
2	153751	0.098209	9.820894	269272	0.171998	17.199834
3	390282	0.249294	24.929386	708681	0.452672	45.267222
4	31830	0.020332	2.033151	306466	0.195756	19.575612



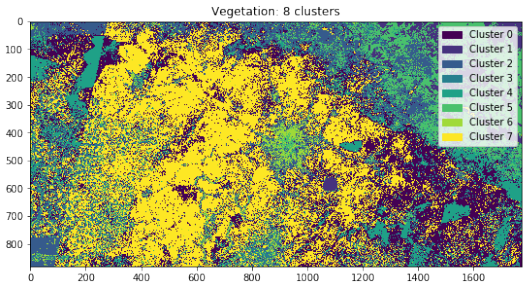


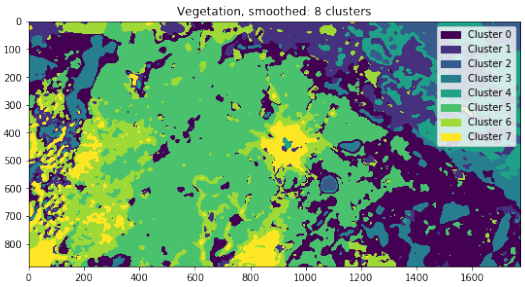
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	298178	0.190462	19.046214	636101	0.406312	40.631152
1	28011	0.017892	1.789211	164926	0.105347	10.534700
2	221473	0.141467	14.146658	107816	0.068868	6.886781
3	167492	0.106986	10.698604	282560	0.180486	18.048609
4	770632	0.492244	49.224362	81783	0.052239	5.223915
5	79764	0.050950	5.094951	292364	0.186748	18.674843



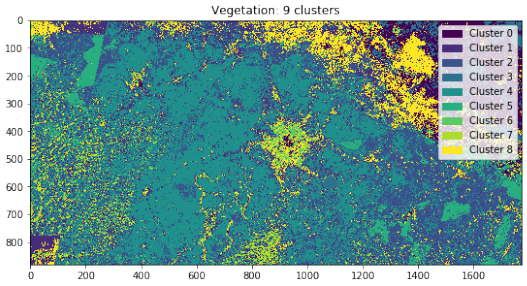


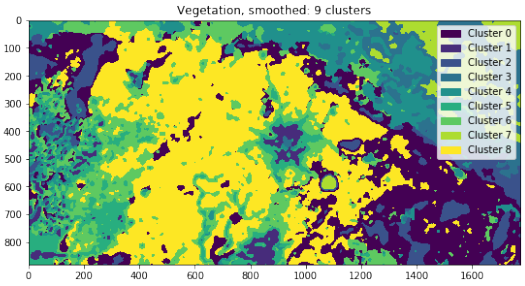
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	173359	0.110734	11.073361	253653	0.162022	16.202165
1	26510	0.016933	1.693335	137357	0.087737	8.773722
2	285127	0.182126	18.212577	280502	0.179172	17.917154
3	89164	0.056954	5.695379	84638	0.054063	5.406279
4	218980	0.139874	13.987417	124715	0.079662	7.966210
5	58294	0.037235	3.723548	76979	0.049171	4.917058
6	714116	0.456144	45.614385	607706	0.388174	38.817412





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	363411	0.232130	23.212992	318864	0.203675	20.367539
1	76941	0.049146	4.914631	138757	0.088631	8.863147
2	199255	0.127275	12.727476	76700	0.048992	4.899237
3	26278	0.016785	1.678516	94834	0.060576	6.057552
4	145204	0.092750	9.274951	121662	0.077712	7.771199
5	159244	0.101718	10.171761	514240	0.328472	32.847242
6	55266	0.035301	3.530133	222520	0.142135	14.213535
7	539951	0.344895	34.489540	77973	0.049805	4.980550





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	136850	0.087413	8.741337	287951	0.183930	18.392961
1	144387	0.092228	9.222765	34804	0.022231	2.223116
2	349301	0.223117	22.311712	91367	0.058361	5.836096
3	57368	0.036644	3.664399	121932	0.077884	7.788445
4	520473	0.332454	33.245377	139520	0.089119	8.911884
5	129387	0.082646	8.264635	104152	0.066527	6.652742
6	48638	0.031068	3.106768	236383	0.150990	15.099039
7	25711	0.016423	1.642298	76835	0.049079	4.907860
8	153435	0.098007	9.800709	472606	0.301879	30.187857

3.2.1 Results

A similar story to before with 4 and 5 clusters picking out the city from the surrounding grass/farmland but getting it confused with either or both the mountains and clouds

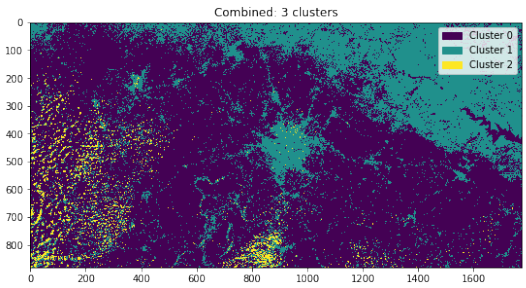
3.3 Combining the images

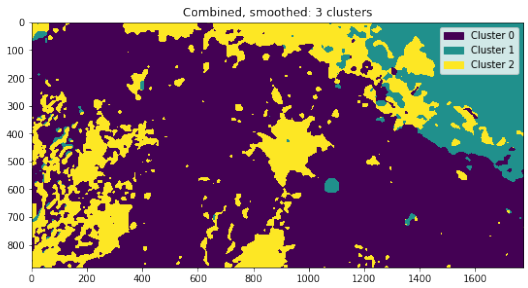
In machine learning, more data is generally better *if* the data is relevant to the problem you are trying to solve. Since all the images appear to go some way to helping us, it makes sense to combine the data and repeat the k-means.

```
In [46]: all_img = np.concatenate([images["Natural"],
                                   images["Vegetation"],
                                   images["Urban"],
                                   images["Agricultural"]
                                  ],
                                  axis = 2
                                 )

         images["Combined"] = all_img

         all_smooth_img = np.concatenate([smooth_images["Natural"],
                                          smooth_images["Vegetation"],
                                          smooth_images["Urban"],
                                          smooth_images["Agricultural"]
                                         ],
                                         axis = 2
                                        )

         smooth_images["Combined"] = all_smooth_img
```

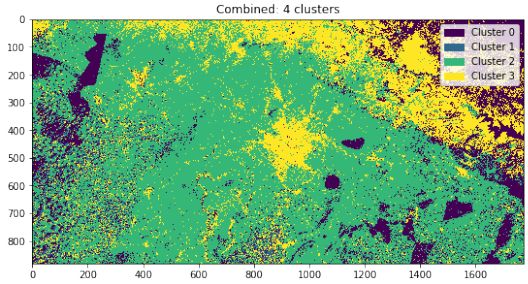
```
In [47]: combo_results = cluster_ks(images["Combined"], range(3,10))
         smooth_combo_results = cluster_ks(smooth_images["Combined"], range(3,10))
```

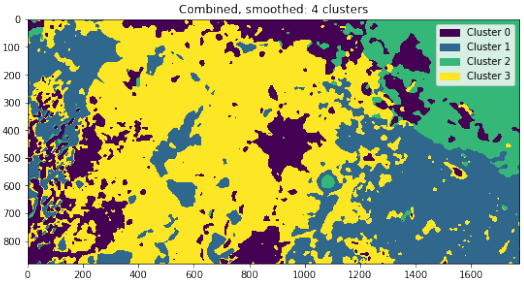
```
In [184]: plt_results(ks = range(3,10),
                      imgs = combo_results,
                      smoothed_imgs = smooth_combo_results,
                      img_name = "Combined"
                     )
```



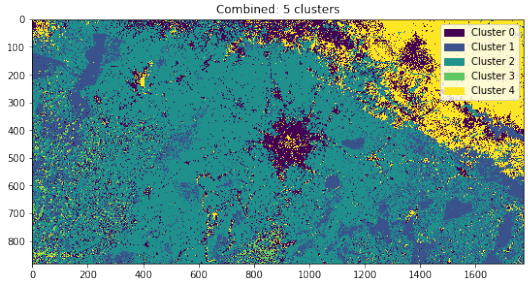


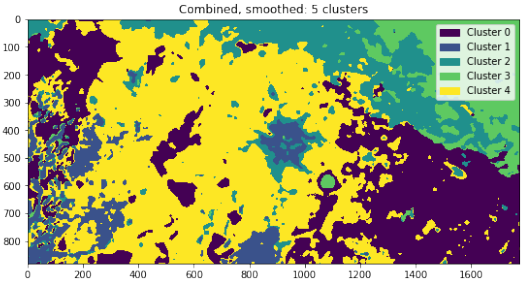
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	1081042	0.690519	69.051899	1030320	0.658120	65.812015
1	447940	0.286123	28.612309	210746	0.134615	13.461467
2	36568	0.023358	2.335793	324484	0.207265	20.726518





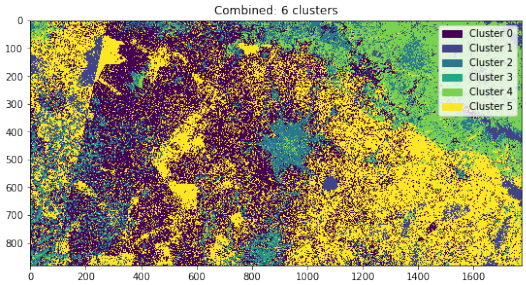
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	268341	0.171404	17.140366	256913	0.164104	16.410399
1	34557	0.022073	2.207339	415854	0.265628	26.562805
2	930235	0.594191	59.419054	215308	0.137529	13.752866
3	332417	0.212332	21.233241	677475	0.432739	43.273929

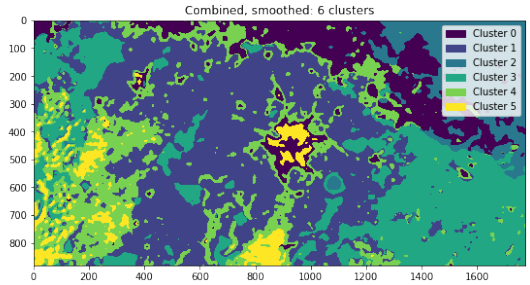




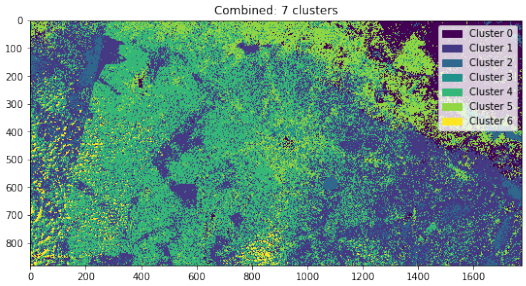
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	198805	0.126987	12.698732	411123	0.262606	26.260611
1	253685	0.162042	16.204209	104947	0.067035	6.703523

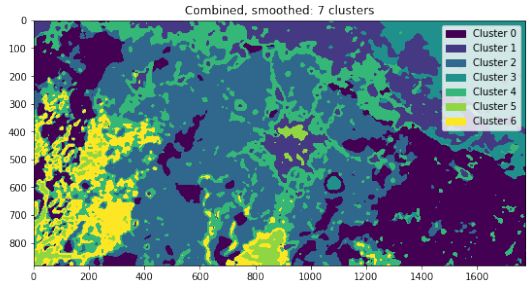
2	853550	0.545208	54.520775	230680	0.147348	14.734758
3	32027	0.020457	2.045735	166468	0.106332	10.633196
4	227483	0.145305	14.530548	652332	0.416679	41.667912





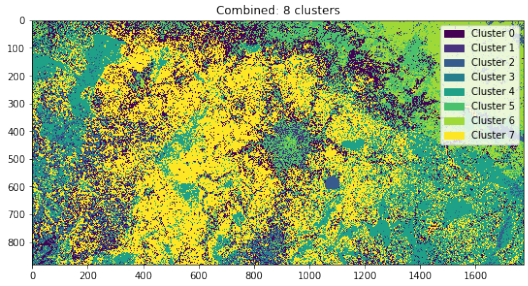
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	517799	0.330746	33.074574	174761	0.111629	11.162914
1	104942	0.067032	6.703203	537296	0.343200	34.319951
2	148598	0.094917	9.491744	162903	0.104055	10.405481
3	30976	0.019786	1.978602	353977	0.226104	22.610393
4	243820	0.155741	15.574079	274763	0.175506	17.550573
5	519415	0.331778	33.177797	61850	0.039507	3.950688

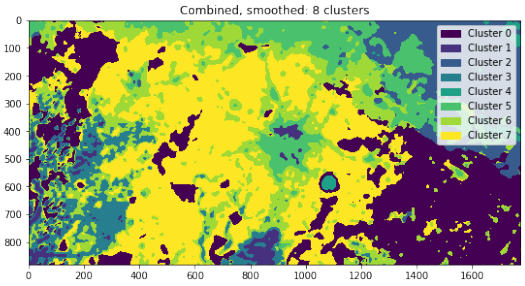




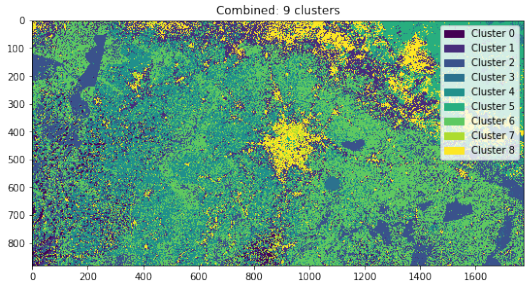
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	171113	0.109299	10.929897	341445	0.218099	21.809907
1	472631	0.301895	30.189454	169010	0.107956	10.795567
2	88057	0.056247	5.624669	467118	0.298373	29.837310
3	58225	0.037191	3.719140	165371	0.105631	10.563125
4	535593	0.342112	34.211172	287588	0.183698	18.369774
5	212776	0.135911	13.591134	31309	0.019999	1.999872

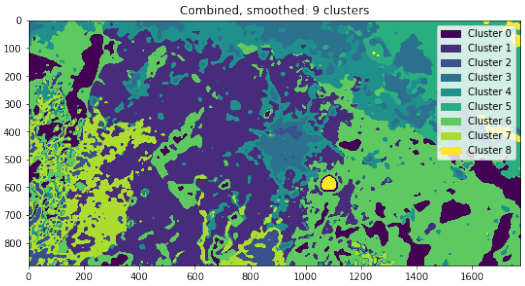
6	27155	0.017345	1.734534	103709	0.066244	6.624445





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	255658	0.163302	16.330235	342251	0.218614	21.861391
1	39512	0.025238	2.523841	29936	0.019122	1.912171
2	53042	0.033881	3.388074	169503	0.108271	10.827058
3	25562	0.016328	1.632781	102573	0.065519	6.551883
4	355903	0.227334	22.733416	11119	0.007102	0.710230
5	136187	0.086990	8.698988	155254	0.099169	9.916898
6	170719	0.109047	10.904730	287464	0.183619	18.361854
7	528967	0.337879	33.787934	467450	0.298585	29.858516





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	25075	0.016017	1.601674	102477	0.065458	6.545751
1	214297	0.136883	13.688288	424250	0.270991	27.099103
2	172573	0.110232	11.023155	31813	0.020321	2.032065
3	39141	0.025001	2.500144	123153	0.078664	7.866437
4	342517	0.218784	21.878381	202958	0.129640	12.964006
5	165049	0.105426	10.542557	170039	0.108613	10.861295

```
6            452295  0.288905  28.890486      374444  0.239177  23.917729
7             35358  0.022585   2.258503      125610  0.080234   8.023378
8            119245  0.076168   7.616812       10806  0.006902   0.690237
```

3.3.1 Results

It is a similar story for our combined image dataset. Four clusters show a nice clear and distinct city in the centre. Unfortunatly it still is getting clustered with both clouds and mountains.

Although the combined image in our data did not appear to give any advantages when it might be expected that adding more features should provide more distict clusters. It maybe that using a brute force approach of putting everything together is not partiularly advantagous as while one or two extra features may help in seperating clusters, the majority may not and just be adding noise to the data.

Using a wide spectra range may still prove useful, we just have to be more selective and avoid multcolinearity.

3.4 Selecting spectra to cluster across

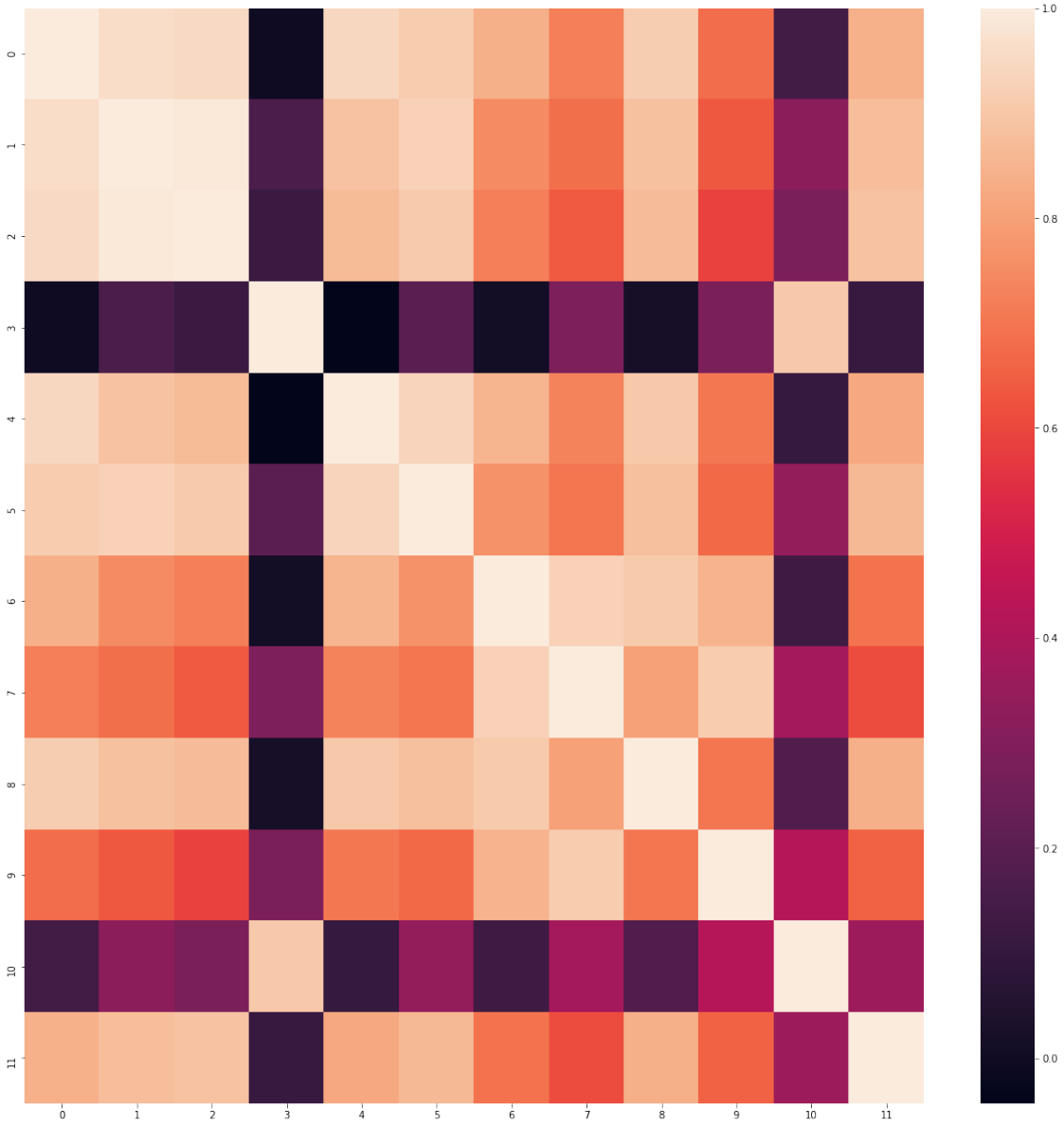
Here we reshape our combined image and then examine correlations between the 12 different features from our the images.

```
In [192]: dims = np.shape(images["Combined"])

          print("image dimensions: {}".format(dims))
          comb_img_matrix = np.reshape(images["Combined"], (dims[0] * dims[1], dims[2]))
          print("flattened image dimensions: {}".format(np.shape(comb_img_matrix)))
```

```
image dimensions: (882, 1775, 12)
flattened image dimensions: (1565550, 12)
```

```
In [196]: corr = np.corrcoef(comb_img_matrix.T)
          sns.heatmap(corr)
```

```
Out[196]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6412e7a400>
```




```
In [243]: corr[:3].sum(axis = 0)

Out[243]: array([2.90979318, 2.95267253, 2.93902626, 0.27261572, 2.70327353,
                 2.75015114, 2.31406399, 2.04566373, 2.66888448, 1.90467709,
                 0.74743387, 2.60777082])
```

It looks like if we combine the natural image (0-2) with the agricultural, that might give us better segmentation as they appear decorrelated with eachother.

We could also try the natural and vegetation (3-5) as vegetation make use of infrared.

Finally, we could selectivley pick the columns that are most decorrelated with the natural image (3, 9 and 10). to see if that helps us

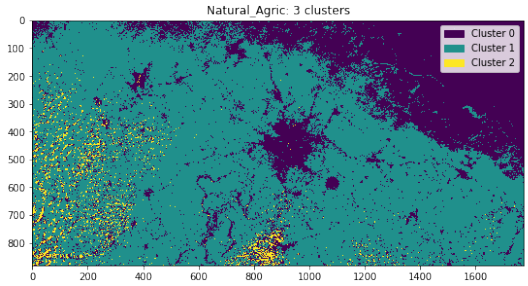
3.5 Natural and Agric images combined

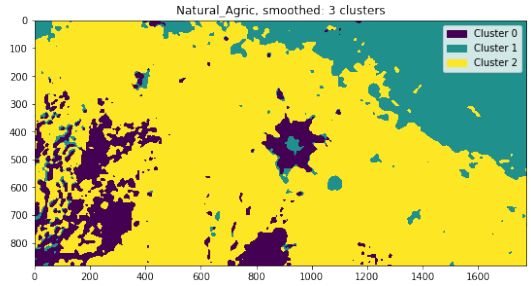
```
In [229]: images["Natural_Agric"]  = np.concatenate([images["Natural"],
                                     images["Agricultural"]
                                    ],
                                    axis = 2
                                   )
```

```
          smooth_images["Natural_Agric"] = np.concatenate([smooth_images["Natural"],
                                           smooth_images["Agricultural"]
                                          ],
                                          axis = 2
                                         )
```

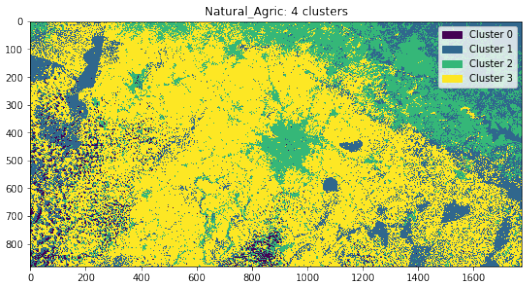
```
In [230]: nat_agric_results = cluster_ks(images["Natural_Agric"], range(3,10))
          nat_agric_combo_results = cluster_ks(smooth_images["Natural_Agric"], range(3,10))
```

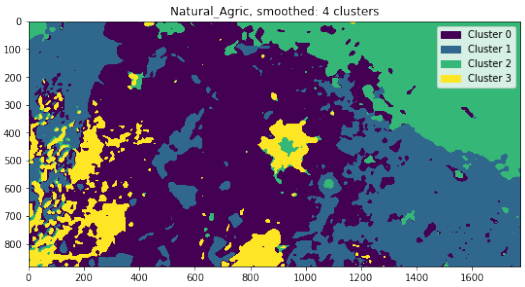
```
In [236]: plt_results(ks = range(3,10),
                      imgs = nat_agric_results ,
                      smoothed_imgs = nat_agric_combo_results,
                      img_name = "Natural_Agric"
                     )
```



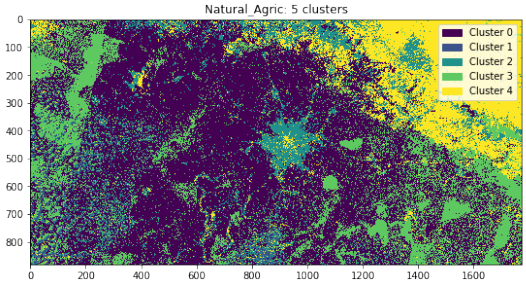


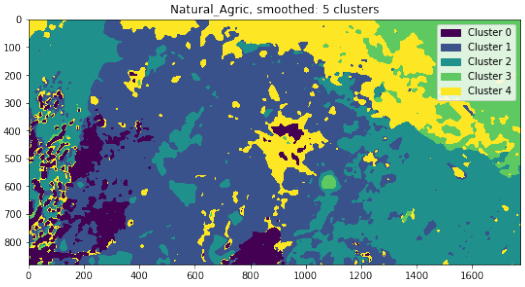
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	448934	0.286758	28.675801	158296	0.101112	10.111207
1	1080743	0.690328	69.032800	339497	0.216855	21.685478
2	35873	0.022914	2.291399	1067757	0.682033	68.203315



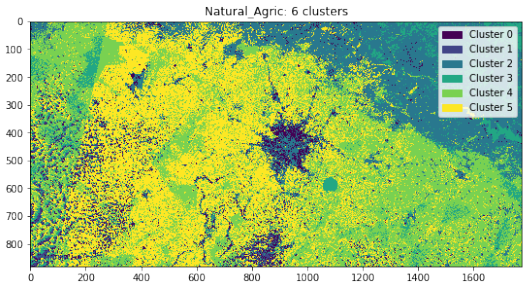


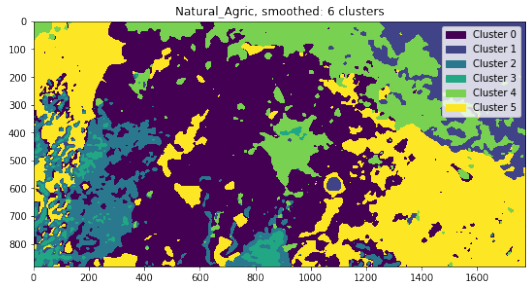
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	34087	0.021773	2.177318	682779	0.436127	43.612724
1	328975	0.210134	21.013382	437337	0.279350	27.935039
2	331189	0.211548	21.154802	319997	0.204399	20.439909
3	871299	0.556545	55.654498	125437	0.080123	8.012328



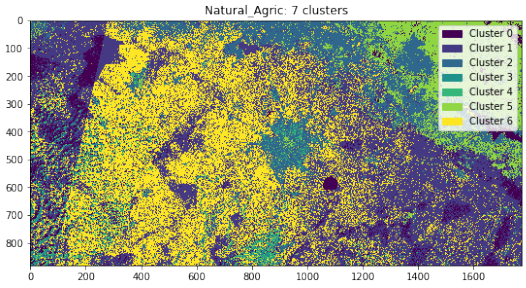


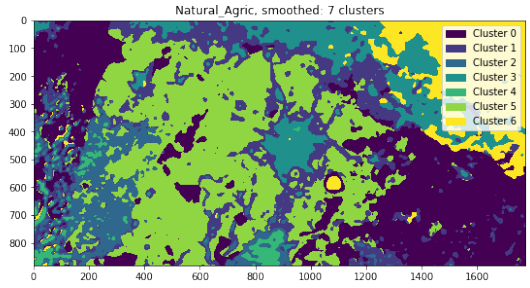
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	793603	0.506916	50.691642	99920	0.063824	6.382422
1	31613	0.020193	2.019290	619362	0.395619	39.561943
2	171423	0.109497	10.949698	427519	0.273079	27.307911
3	312859	0.199840	19.983967	171789	0.109731	10.973077
4	256052	0.163554	16.355402	246960	0.157746	15.774648



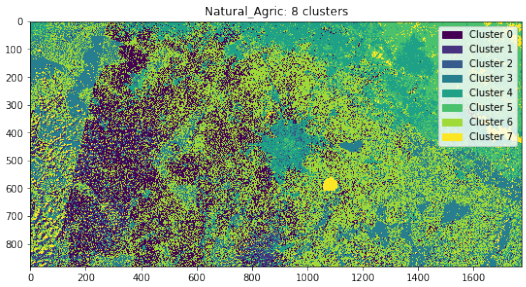


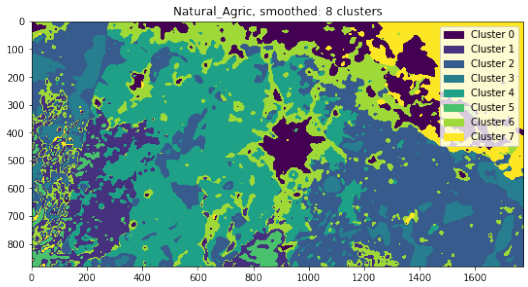
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	59241	0.037840	3.784038	590559	0.377221	37.722142
1	27150	0.017342	1.734215	171176	0.109339	10.933921
2	324540	0.207301	20.730095	138041	0.088174	8.817412
3	125167	0.079951	7.995082	33564	0.021439	2.143911
4	521772	0.333284	33.328351	225262	0.143887	14.388681
5	507680	0.324282	32.428220	406948	0.259939	25.993932



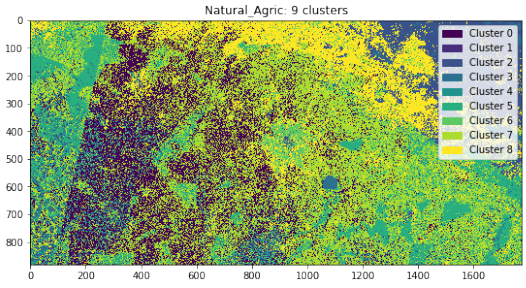


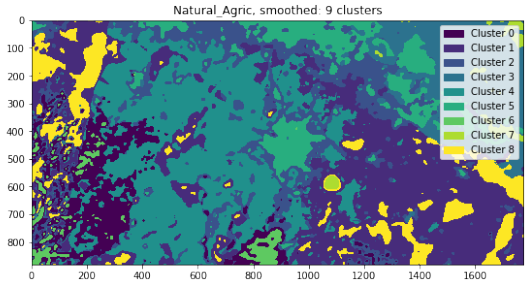
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	99842	0.063774	6.377439	377515	0.241139	24.113890
1	533916	0.341041	34.104053	252211	0.161101	16.110057
2	225753	0.144200	14.420044	119228	0.076157	7.615726
3	24987	0.015961	1.596053	160972	0.102821	10.282137
4	37486	0.023944	2.394430	26806	0.017122	1.712242
5	172977	0.110490	11.048960	457817	0.292432	29.243205
6	470589	0.300590	30.059021	171001	0.109227	10.922743





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	301197	0.192391	19.239053	143110	0.091412	9.141196
1	24618	0.015725	1.572483	116858	0.074643	7.464342
2	35434	0.022634	2.263358	398225	0.254367	25.436747
3	240675	0.153732	15.373192	87459	0.055865	5.586471
4	211826	0.135305	13.530453	395654	0.252725	25.272524
5	176098	0.112483	11.248315	25846	0.016509	1.650921
6	523389	0.334316	33.431637	224878	0.143642	14.364153
7	52313	0.033415	3.341509	173520	0.110836	11.083645





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	305113	0.194892	19.489189	116972	0.074716	7.471623
1	19188	0.012256	1.225640	391517	0.250083	25.008272
2	156265	0.099815	9.981476	220264	0.140694	14.069432
3	48901	0.031236	3.123567	175606	0.112169	11.216889
4	21105	0.013481	1.348089	388812	0.248355	24.835489
5	232082	0.148243	14.824311	125646	0.080257	8.025678
6	65853	0.042064	4.206381	25733	0.016437	1.643703
7	503544	0.321640	32.164032	13634	0.008709	0.870876
8	213499	0.136373	13.637316	107366	0.068580	6.858037

3.5.1 Results

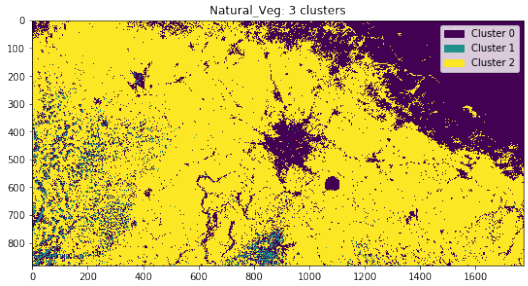
We can see from the unsmoothed images, that there is less noise in the grouping for this which shows an improvement. Unfortunately, we still get the algorithim grouping the city in with clouds or the mountains.

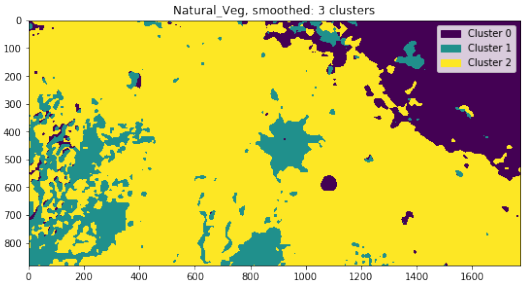
3.6 Natural and vegetation images combined

```
In [233]: images["Natural_Veg"]  = np.concatenate([images["Natural"],
                                                   images["Vegetation"]
                                                   ],
                                                   axis = 2
                                                  )


          smooth_images["Natural_Veg"] = np.concatenate([smooth_images["Natural"],
                                                         smooth_images["Vegetation"]
                                                        ],
                                                        axis = 2
                                                       )
```

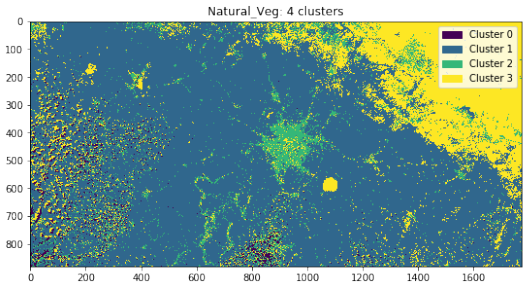
```
In [237]: nat_veg_results = cluster_ks(images["Natural_Veg"], range(3,10))
          nat_veg_combo_results = cluster_ks(smooth_images["Natural_Veg"], range(3,10))
```

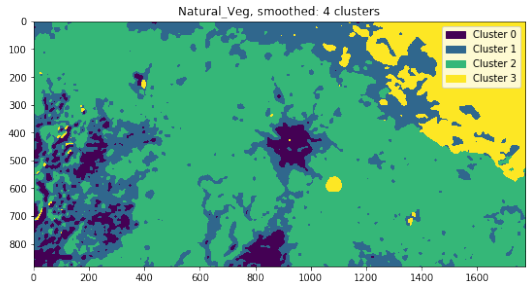
```
In [238]: plt_results(ks = range(3,10),
                      imgs = nat_veg_results,
                      smoothed_imgs = nat_veg_combo_results,
                      img_name = "Natural_Veg"
                     )
```



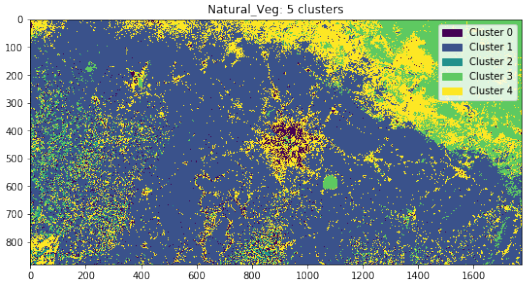


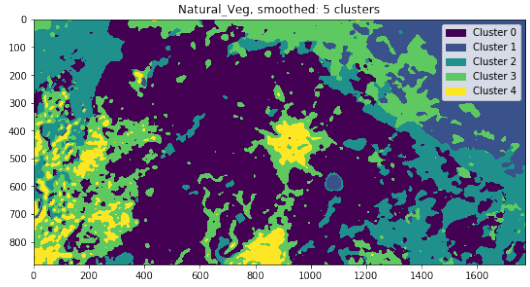
	unsmoothed			smoothed		
cluster	pixel_count	ratio	% total	pixel_count	ratio	% total
0	413628	0.264206	26.420619	264967	0.169249	16.924851
1	36085	0.023049	2.304941	185197	0.118295	11.829517
2	1115837	0.712744	71.274440	1115386	0.712456	71.245633



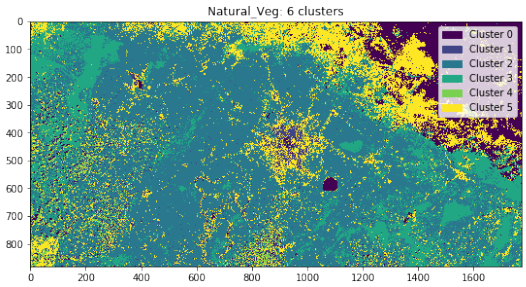


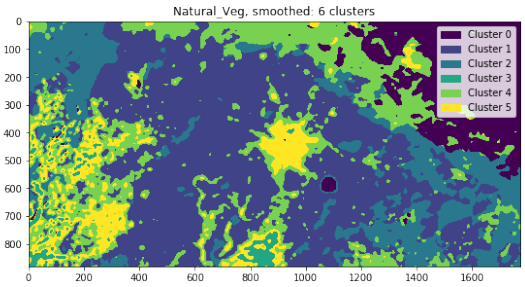
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	29409	0.018785	1.878509	79038	0.050486	5.048577
1	1102140	0.703995	70.399540	349669	0.223352	22.335218
2	122327	0.078137	7.813676	927662	0.592547	59.254703
3	311674	0.199083	19.908275	209181	0.133615	13.361502



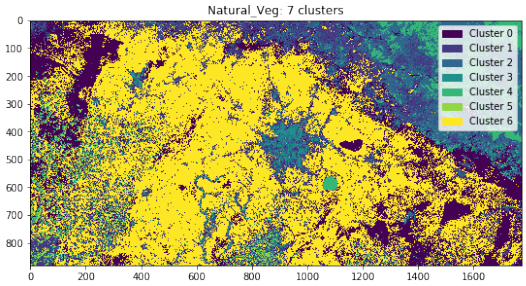


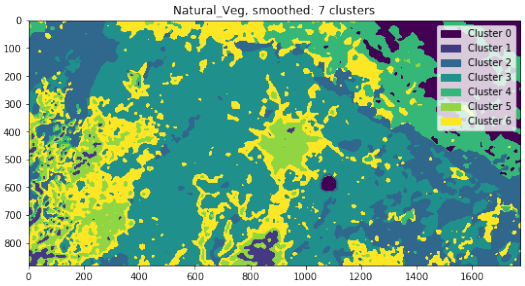
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	53138	0.033942	3.394207	681691	0.435432	43.543228
1	1003018	0.640681	64.068091	203723	0.130129	13.012871
2	26012	0.016615	1.661525	280479	0.179157	17.915685
3	222009	0.141809	14.180895	324844	0.207495	20.749513
4	261373	0.166953	16.695283	74813	0.047787	4.778704



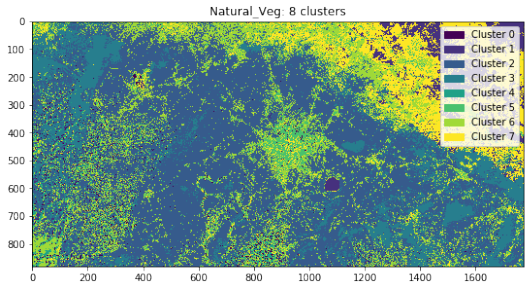


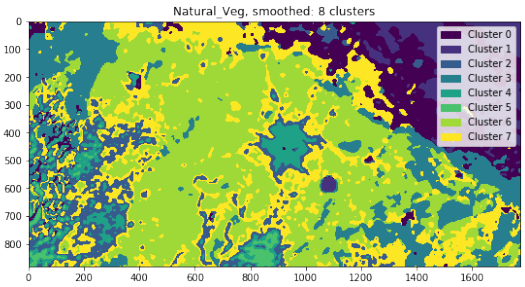
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	189320	0.120929	12.092875	205556	0.131300	13.129954
1	51155	0.032675	3.267542	654658	0.418165	41.816486
2	768974	0.491185	49.118457	255772	0.163375	16.337517
3	289508	0.184924	18.492415	21068	0.013457	1.345725
4	25819	0.016492	1.649197	319439	0.204043	20.404267
5	240774	0.153795	15.379515	109057	0.069661	6.966050



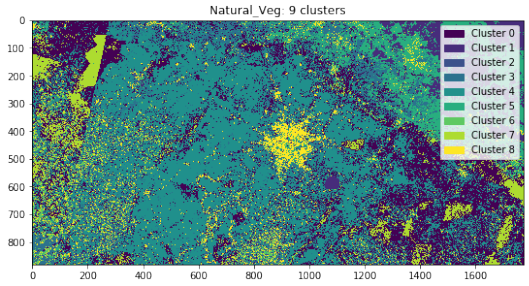


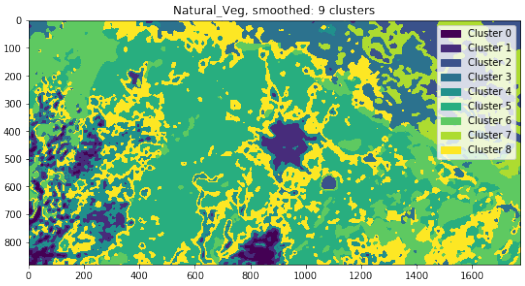
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	293646	0.187567	18.756731	124629	0.079607	7.960717
1	196008	0.125201	12.520073	21178	0.013528	1.352751
2	181069	0.115658	11.565839	253986	0.162234	16.223436
3	50747	0.032415	3.241481	603876	0.385728	38.572770
4	84363	0.053887	5.388713	165115	0.105468	10.546773
5	25777	0.016465	1.646514	109063	0.069664	6.966434
6	733940	0.468806	46.880649	287703	0.183771	18.377120





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	19875	0.012695	1.269522	166632	0.106437	10.643672
1	93045	0.059433	5.943279	125295	0.080033	8.003258
2	681925	0.435582	43.558174	132330	0.084526	8.452620
3	262230	0.167500	16.750024	215350	0.137555	13.755549
4	21727	0.013878	1.387819	66793	0.042664	4.266424
5	70584	0.045086	4.508575	15276	0.009758	0.975759
6	229818	0.146797	14.679697	532074	0.339864	33.986395
7	186346	0.119029	11.902910	311800	0.199163	19.916323





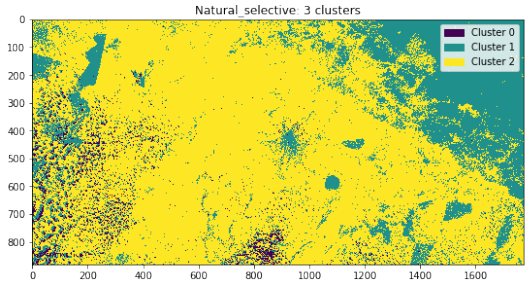
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	338455	0.216189	21.618920	15253	0.009743	0.974290
1	94386	0.060289	6.028936	66167	0.042264	4.226438
2	19202	0.012265	1.226534	77062	0.049224	4.922360
3	191251	0.122162	12.216218	133698	0.085400	8.540002
4	554119	0.353945	35.394526	120964	0.077266	7.726614
5	185108	0.118238	11.823832	503132	0.321377	32.137715
6	21574	0.013780	1.378046	215495	0.137648	13.764811
7	95147	0.060775	6.077545	122219	0.078068	7.806777
8	66308	0.042354	4.235444	311560	0.199010	19.900993

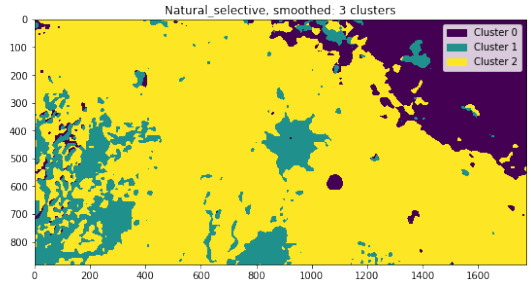
3.6.1 Results

In the three cluster results, we see no noise, but we lose some fine detail, like where the main roads are in this image. Other than than, it is the same again; struggling to sepearte city from mountians and clouds

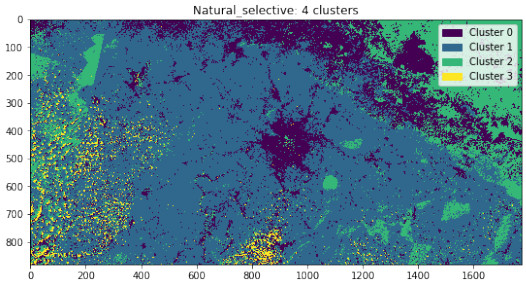
3.7 Selective decorrelated columns with Natural image

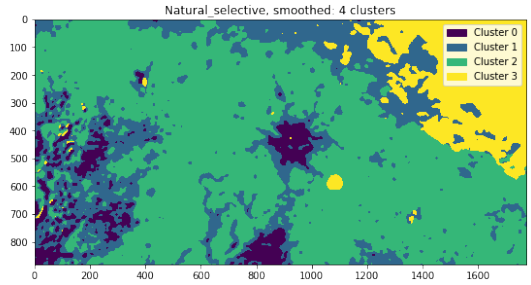
```
In [269]: images["Natural_selective"] = images["Combined"][:,:, [1,2,3,4,9,10]]

In [271]: smooth_images["Natural_selective"] = smooth_images["Combined"][:,:, [1,2,3,4,9,10]]

In [270]: images["Natural_selective"].shape

Out[270]: (882, 1775, 6)

In [272]: smooth_images["Natural_selective"].shape

Out[272]: (882, 1775, 6)

In [273]: nat_sel_results = cluster_ks(images["Natural_selective"], range(3,10))
          nat_sel_combo_results = cluster_ks(smooth_images["Natural_selective"], range(3,10))

In [275]: plt_results(ks = range(3,10),
                      imgs = nat_sel_results,
                      smoothed_imgs = nat_veg_combo_results,
                      img_name = "Natural_selective"
                     )
```



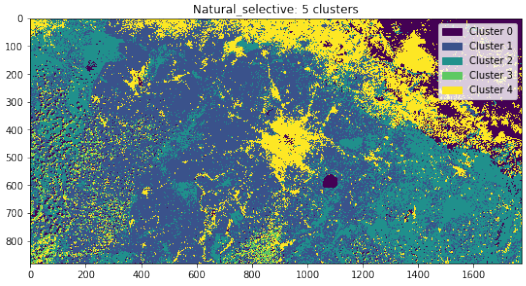


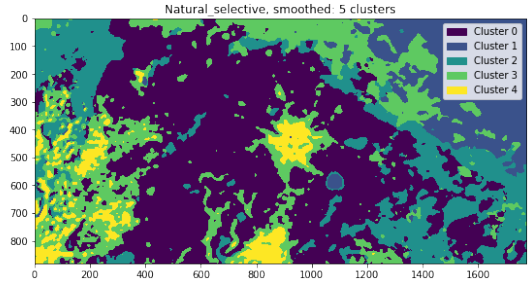
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	40156	0.025650	2.564977	264967	0.169249	16.924851
1	394403	0.251926	25.192616	185197	0.118295	11.829517
2	1130991	0.722424	72.242407	1115386	0.712456	71.245633



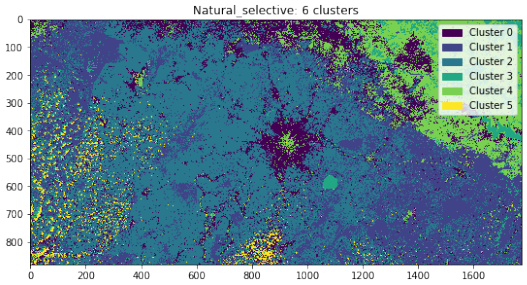


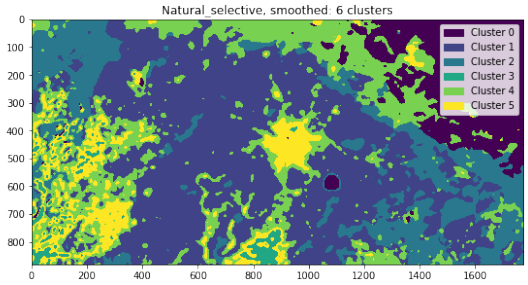
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	279537	0.178555	17.855514	79038	0.050486	5.048577
1	961493	0.614157	61.415669	349669	0.223352	22.335218
2	289272	0.184773	18.477340	927662	0.592547	59.254703
3	35248	0.022515	2.251477	209181	0.133615	13.361502



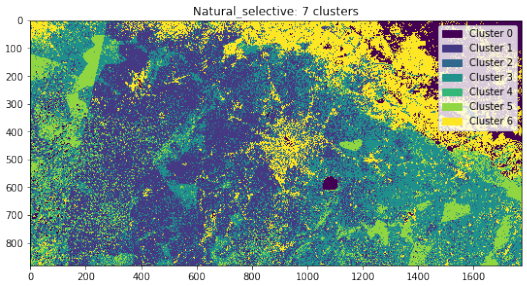


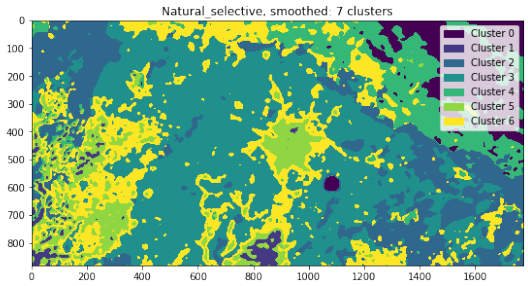
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	190180	0.121478	12.147807	681691	0.435432	43.543228
1	653905	0.417684	41.768388	203723	0.130129	13.012871
2	421210	0.269049	26.904922	280479	0.179157	17.915685
3	35482	0.022664	2.266424	324844	0.207495	20.749513
4	264773	0.169125	16.912459	74813	0.047787	4.778704



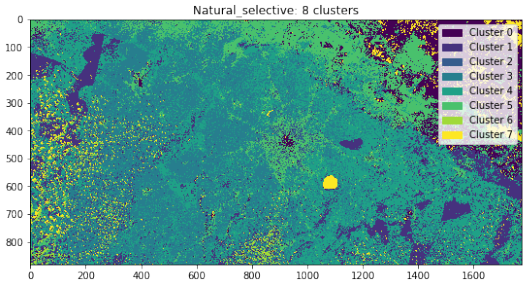


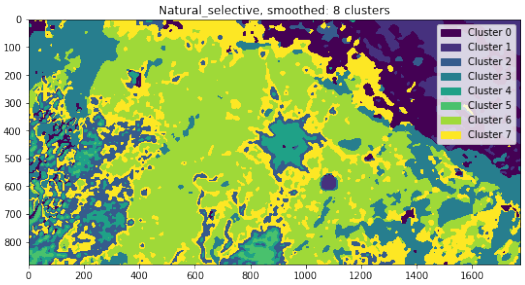
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	190968	0.121981	12.198141	205556	0.131300	13.129954
1	421536	0.269257	26.925745	654658	0.418165	41.816486
2	642529	0.410417	41.041743	255772	0.163375	16.337517
3	77912	0.049767	4.976654	21068	0.013457	1.345725
4	199218	0.127251	12.725113	319439	0.204043	20.404267
5	33387	0.021326	2.132605	109057	0.069661	6.966050



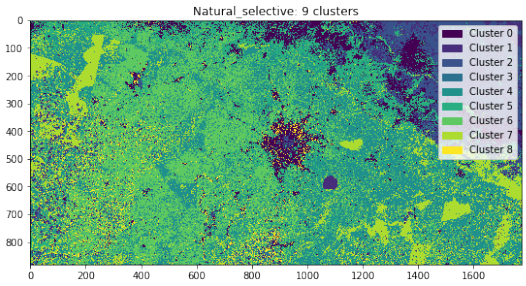


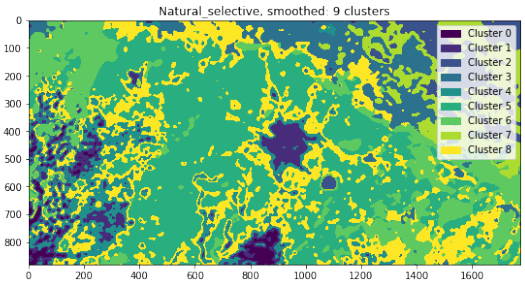
	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	140493	0.089740	8.974035	124629	0.079607	7.960717
1	472315	0.301693	30.169270	21178	0.013528	1.352751
2	57742	0.036883	3.688288	253986	0.162234	16.223436
3	474259	0.302934	30.293443	603876	0.385728	38.572770
4	27736	0.017716	1.771646	165115	0.105468	10.546773
5	142841	0.091240	9.124014	109063	0.069664	6.966434
6	250164	0.159793	15.979304	287703	0.183771	18.377120





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	157786	0.100786	10.078631	166632	0.106437	10.643672
1	174499	0.111462	11.146179	125295	0.080033	8.003258
2	47859	0.030570	3.057009	132330	0.084526	8.452620
3	414189	0.264565	26.456453	215350	0.137555	13.755549
4	474929	0.303362	30.336240	66793	0.042664	4.266424
5	218415	0.139513	13.951327	15276	0.009758	0.975759
6	26840	0.017144	1.714413	532074	0.339864	33.986395
7	51033	0.032597	3.259749	311800	0.199163	19.916323





	unsmoothed			smoothed		
	pixel_count	ratio	% total	pixel_count	ratio	% total
cluster						
0	116768	0.074586	7.458593	15253	0.009743	0.974290
1	51109	0.032646	3.264603	66167	0.042264	4.226438
2	157917	0.100870	10.086998	77062	0.049224	4.922360
3	25801	0.016480	1.648047	133698	0.085400	8.540002
4	426047	0.272139	27.213886	120964	0.077266	7.726614
5	200415	0.128016	12.801571	503132	0.321377	32.137715
6	376224	0.240314	24.031427	215495	0.137648	13.764811
7	173513	0.110832	11.083198	122219	0.078068	7.806777
8	37756	0.024117	2.411676	311560	0.199010	19.900993

In the unsmoothed data we almost entirely lose the city with three clusters, but the smoothed data looks better.

Again we see the same issues of grouping the city, mountains and clouds as previous images.

3.8 Discussion

All our images show similar results; 4-6 clusters appears to provide the best segmentation. However, with the images provided K-means often groups in urban areas with the more mountainous areas in the north and east as well as the clouds in the image to the west and south. There is also the issue of the image being made up of two sepearte photographs, visable from the divide in shading in the top left of the image.

These problems could be solved by aquiring more suitable images. A cloudless, colour adjusted composite image would appear to be more ideal for this. An issue might be timeframe: a composite image may be made up of photographs taken across different days. When trying to measure change in land use this could prove problematic if the image is taken over a long peroid of time, so when using a composite image caution needs to be taken when considering the data collection timeframe.

A different approach could be to used supervised learning. To do this we will have to get a labelled dataset. To get the labels, we could align a satellite image with a shapefile to get GIS information about each of the pixels i.e does a pixel fall within the boundary of an urban area like a town or a city. You can then use this data to train a classification algorithm to identify urban and non-urban areas. With a timeline of satellite imagery, it would be possible to run the same model across images then compare the change in classifications to get change of different areas over time.

If GIS information is not available. With good clustering, this could provide a labelled dataset in place of the shapefile, though the former method is preferable as it is will be less prone to error.

3.9 Summary

- Smoothed images appear better for gross segmentation
- Clouds get grouped in with terrian, ideally any images will need to be free of clouds.
- Unsupervised methods may be of some use. If images can come with gis data, it may be possible to use this information as labels in a supervised machine learning approach which could be more useable.
- With an improved clustering or GIS data, it may be possible to detect change in land use over time.