Assignment 4 - Machine Learning

G.J. Leenen (476908) , W.M. Brus (484505) and C.A. Vriends (440435)

October 8, 2019

Throughout this assignment, we will interpret the dimensions of the source dataset ($600 \times 600 \times 3$) as representing 600^2 observations of pixels, where each pixel has 3 dimensions (R, G, B).

Part A

In this part we use K-means clustering method to cluster the sample pixels. In doing so, we partition all observations into K clusters. The goal is to assign each observation to the cluster with the closest mean. If we define z_{ik} in the following manner:

$$z_{ik} = \begin{cases} 1, & \text{if point } i \text{ is in cluster } k, \\ 0, & \text{otherwise,} \end{cases}$$

then the K-means algorithm attempts to solve the following optimisation problem:

$$\begin{array}{ll} \min\limits_{\mu,z} & \sum_{k=1}^{K} \sum_{i=1}^{N} d(x_i, \mu_k) z_{ik} \\ \text{s.t.} & \sum_{k=1}^{K} z_{ik} = 1,\ i = 1, 2, \dots, N; \\ & z_{ik} \in \{0, 1\},\ i = 1, 2, \dots, N,\ k = 1, 2, \dots, K; \\ & \mu_k \in \mathbb{R}^p,\ k = 1, 2, \dots, K, \end{array} \tag{1}$$

where x_i is a given observation, μ_k is the centroid of cluster k, and $d(a, b)$ denotes the dissimilarity between any two points a and b. The question is now how to measure dissimilarity. Generally, there are two different groups of measures. The first group is the set of *distance-based measures*, such as Euclidean or Manhattan distance, while the second group is the set of *correlation-based measures*. As explained in James, Witten, Hastie, and Tibshirani, 2014, correlation-based measures may cluster observations that have high correlation even though they are dissimilar in terms of distance-based dissimilarity (e.g. Euclidean distance). As our dataset originates from an image of a cell, we argue that one can meaningfully differentiate between observations based on distance in RGB space; if we assume that similar parts of the cell have similar colors, then the clustering algorithms should be able to effectively cluster pixels based on how far apart their RGB values are. We therefore opt for a distance-based dissimilarity measure, and use Euclidean distance as all dimensions of RGB space are measured on the same scale (0-255). Of course, our reasoning is valid only based on the premise that similar cell components have similar colors, and domain knowledge is necessary to justify this assumption.

We perform our analysis using the `KMeans()` method of `sklearn`'s `cluster` class. Aside from the number of clusters to form (K) and the number of runs to do in parallel, we stick with default settings, which means the following. The K-means algorithm is run with 10 different centroid seeds, and the results produced are those of the run that led to the lowest cluster error (as given in eq. (1), in *sklearn* this is referred to as *inertia*). Centroids seeds are not chosen fully at random, but rather through a procedure called *kmeans++*, which has been shown to lead to more robust (i.e., lower variance in inertia) clustering. Further, two stopping conditions are imposed on the algorithm, both of which are individually sufficient to break out of the loop: a max of 300 iterations, and a tolerance for declaring convergence (based on changes in inertia) of 10^{-4}. Lastly, we let `KMeans()` automatically determine the specific K-means algorithm to use, though in practice this means we employ Elkan's algorithm as our data is dense.

Note that the settings given above imply that the algorithm must necessarily improve the objective at each step it takes. Upon initialisation, observations are assigned (quasi-) randomly to a cluster, such that the initial solution is a feasible solution. Not taking a step is therefore always an option for the algorithm, since the solution at the start of any step is already in the feasible set, and convergence (to a tolerance) is a stopping condition. Taking a step thus necessarily means the algorithm has improved the objective.

The results can be found in table 1. We see that the more clusters we allow for, the lower is our cluster error. This was to be expected, as more clusters should allow the algorithm to find a more similar cluster mean, reducing distances and thus improving the objective. There are also two clear downsides to choosing a large k, the first of which is the computational burden. Higher values of k are associated with a large burden and thereby a greater running time. The second downside is the interpretability of the output. To see why, consider letting $k \to N$. The larger the number of clusters, the closer we are to the situation prior to clustering, and the harder it becomes to determine what separates one cluster from another. There is a fine balance between speed and interpretability vs. improving the objective.

Table 1: Results K-Means Clustering

	$K=2$		$K=4$		$K=8$		$K=16$	
k	Size $\times 10^4$	Centroid*	Size $\times 10^4$	Centroid*	Size $\times 10^4$	Centroid*	Size $\times 10^4$	Centroid*
1	14.21	[129 85 131]	5.71	[91 55 107]	5.67	[216 192 201]	3.26	[221 208 210]
2	21.79	[217 198 202]	12.23	[226 223 215]	3.34	[103 63 116]	1.85	[197 142 169]
3			8.42	[155 106 147]	4.36	[172 129 164]	1.48	[95 74 135]
4			9.64	[204 166 185]	9.74	[228 229 218]	8.41	[229 231 219]
5					2.02	[66 37 89]	2.54	[215 166 180]
6					2.34	[170 86 122]	1.25	[169 85 121]
7					3.33	[128 100 151]	0.94	[57 30 75]
8					5.20	[203 161 181]	1.96	[166 136 171]
9							2.12	[148 113 158]
10							1.89	[123 95 147]
11							1.74	[187 166 190]
12							1.31	[100 48 94]
13							3.54	[215 186 197]
14							1.39	[189 111 143]
15							1.28	[135 65 108]
16							1.02	[66 51 116]
Inertia	**8.45** $\times 10^8$		**3.04** $\times 10^8$		**1.51** $\times 10^8$		**0.79** $\times 10^8$	

*Centroid coordinates have been rounded to the nearest integer.

Part B

In this part we use the agglomerative hierarchical clustering algorithm to cluster the image pixels. In this method, each observation initially its own cluster. At every iteration, the algorithm proceeds by combining the pair of clusters that are 'closest', and ends when the number of clusters has been reduced to K. We already have a measure dissimilarity between two observations (Euclidean distance), which we will retain here, but agglomerative clustering further requires a definition of cluster dissimilarity. As explained below, we use two such measures, and again run the algorithm for each $K \in \{2, 4, 8, 16\}$. Other than this, we only pass default settings to `sklearn`'s `AgglomerativeClustering()`, i.e. no other notable parameters are set.

According to James et al., 2014, the four most commonly used linkage types are *complete*, *single*, *average* and *centroid*. Linkage generalises the concept of dissimilarity between observations to dissimilarity between groups of observations, i.e. the proximity of two clusters. Complete, single and average linkage respectively use maximum, minimum and average pairwise dissimilarity between clusters, while centroid linkage uses the dissimilarity in centroids of clusters. We add one more agglomeration method to this list, namely Ward's method. Ward's method is a variance-based method, and attempts to combine observations such that the increase in the sum of squared distances is minimal (comparable to K-means). Although each methods has its disadvantages, multiple sources state that the average linkage method and Ward's method are generally

the most accurate [see e.g. Johnson and Wichern, 2002 and Milligan, 1980]. James et al., 2014 similarly states that the average linkage method generally results in the most balanced dendrograms (but does not discuss Ward's method). We therefore use the average linkage method and Ward's method.
Formally, the average linkage method measures the proximity of two clusters C_A and C_B as:

$$proximity_{Average}(C_A, C_B) = \frac{1}{m_A m_B} \times \sum_{x_i \in C_A} \sum_{x_j \in C_B} ||x_i - x_j||^2, \tag{2}$$

where m_k denotes the number of observations in cluster k, and where we use the Euclidian norm to measure dissimilarity between individual observations, for reasons explained in part A. Note that the average linkage is scale-variant, but this is not a concern here since all features of a pixel are measured on the same scale.
Our second agglomeration procedure is Ward's method, which minimises the increase in the sum of squared distances. If we define proximity based on this objective, we get the following:

$$proximity_{Ward}(C_A, C_B) = \sum_{x_i \in C_A \cup C_B} ||x_i - \mu_{C_A \cup C_B}||^2 - \sum_{x_i \in C_A} ||x_i - \mu_{C_A}||^2 - \sum_{x_i \in C_B} ||x_i - \mu_{C_B}||^2, \tag{3}$$

where μ_k is the centroid of cluster k, such that the two 'closest' clusters are those that result in the smallest increase in the sum of squared distances.

Before we can continue to the results, we must address a computational issue. As mentioned, each pixel initially forms its own cluster in the hierarchical method. Consequently, it is not possible to store all the pairwise dissimilarities, as this requires storing $N(N-1)/2 = 64,799,820,000$ (single-precision floating point) numbers, or some 260GB of internal memory. This is not possible for us, which is the computational issue hinted at in the assignment. In other words, the large number of pixels is the bottleneck. There are several potential approaches to reducing this number, of which we discuss two. To see how the approaches might help, consider that an image has two essential dimensions, *resolution* (here, 600×600) and *depth* (3 in this case, RGB). Thus, to reduce the computational burden we should reduce the resolution or flatten the depth of the image (or a combination of the two). Whichever approach is used, it is imperative that the essential features of the image (e.g. sharp edges, stark contrast between pixels) are not lost, such that the clusters still have a sensible interpretation.

The first approach we considered was *max pooling (2D)*, which is often used in convolutional neural networks (CNNs) as mentioned by Goodfellow, Bengio, and Courville, 2016. We do not have space to elaborate fully on the method here, but essentially it reduces image resolution, by grouping neighbouring pixels using a 'filter' that is slid across the image. In this process image depth is maintained, and the number of observations is reduced by 93.75%.

The second approach, the *black and white transform*, is a mixture of flattening the depth and reducing the number of pixels by assuming that pixels with the same RGB values belong to the same cluster. The original image is converted to a grayscale image, which flattens the depth from 3 to 1. Duplicate RGB values are then removed from the grayscale image, thus resulting in a maximum of 256 pixels (as the grayscale values range 0-255). The clustering procedure will now be based on the brightness of the grayscale pixels. Pixels with a similar brightness are clustered together, after which the pixels in the 600×600 **grayscale** image are assigned to the same cluster that their pixel value (within the set of unique values) was assigned to. For our dataset, this approach reduces the number of pixels used in the clustering process by 99.9% (360,000 to 218). The crucial assumption in this approach is that clustering only on pixel brightness captures the meaningful latent structure of the image.

It is difficult to evaluate each approach based on the quality of the clustering. This is an unsupervised learning problem and we do not have access to a comparable, validated clustering (e.g. constructed by a domain expert). For our analysis, we visually inspect the clustered image for the two approaches and (subjectively) evaluate the quality of the clustering. If we compare max pooling (figure 1a) with the black and white transform (figure 1b), it seems that the latter approach results in a more sophisticated clustering. This could suggest that resolution is of greater importance than depth for our image. We therefore opt for the black and white transform, as it drastically reduces the number of pixels and produces a clustering that seems more sophisticated than that of the max pooling approach. In table 2 we see that for both linkages, the sizes of the clusters and the cluster errors decrease again as k increases. Cluster errors have been calculated as in part A, by summing squared within-cluster deviations across all observations. We see that average linkages has slightly lower cluster errors than Ward's method in general. This could be specific to our image.

Table 2: Results Agglomerative Hierarchical Clustering*

	Average linkage								**Ward's method**							
	$K=2$		$K=4$		$K=8$		$K=16$		$K=2$		$K=4$		$K=8$		$K=16$	
k	Size	Mean	Size	Mean	Size	Mean	Size	Mean	Size	Mean	Size	Mean	Size	Mean	Size	Mean
1	13.55	101	9.50	117	5.13	165	3.88	188	28.84	186	13.40	150	4.03	92	3.80	195
2	22.45	202	17.32	213	4.30	100	2.32	156	7.16	77	6.23	82	0.93	42	2.16	100
3			4.05	63	5.20	131	2.82	172			15.44	217	5.41	199	2.48	116
4			5.13	165	3.13	69	2.59	139			0.93	42	10.03	227	0.75	45
5					3.41	203	2.61	124					5.86	173	0.18	30
6					0.93	42	1.33	60					2.20	65	1.33	60
7					10.03	227	1.80	76					3.74	143	3.49	220
8					3.88	188	1.60	207					3.80	120	3.43	180
9							1.98	224							1.87	84
10							2.32	108							2.59	139
11							1.98	92							2.43	164
12							0.75	45							6.54	230
13							0.18	30							1.60	207
14							6.54	230							1.15	151
15							1.51	216							1.31	127
16							1.80	199							0.88	72
In.	2.73×10^8		9.49×10^7		2.00×10^7		5.20×10^6		4.63×10^8		1.25×10^8		1.09×10^7		5.49×10^6	

*Sizes are $\times 10^4$. *Mean* refers to the mean brightness of a cluster. *In.* is the 1-D (grayscale) equivalent of inertia defined in part A.

Part C

Last, we use spectral clustering. The idea of this method is essentially to transform our data in such a way that the K-means clustering method will work very efficiently.

A formal interpretation of the spectral algorithm is that the affinity (or similarities matrix) matrix W represents the weighted relations between all observation pairs, and that an associated similarity graph can be constructed. Each observation (or pixel) is a node and the paired observations that have "enough" similarities (e.g. exceeding a certain threshold) are connected by a weighted edge. According to Hastie, Tibshirani, and Friedman, 2009 this reformulates the general problem of clustering as a graph-partitioning problem. Therein, the objective of spectral clustering is to partition the graph in such a way that the weights or edges (the edges are weighted by the similarities) between local neighbourhoods are low, and within local neighbourhoods are high. The similarity graph in `sklearn` is a fully connected graph. All vertices are interconnected, although some edges are still equal to zero. This is determined by the choice of kernel.

There are different ways to construct an affinity matrix. This depends on the choice of kernel, which determines the pairwise similarity, and in turn its associated similarity graph. In our case, the choice of kernel is limited to the options from the `pairwise_kernels` class from `sklearn`. We opt for the same kernel as in Ng, Jordan, and Weiss, 2002, the radial basis function kernel (RBF). This is defined in the following manner: $K_{RBF}(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$, where $\gamma = \frac{1}{2\sigma^2}$. Gamma is a free parameter which influences the behavior of the pairwise similarity and how quickly it decreases when distance increases. In our implementation we use `sklearn's` default of 1. We define

$$g_i = \sum_{i'} w_{ii'} \tag{4}$$

as the degree of vertex (pixel) i, which equals the sum of the weights of the connected edges to the vertex point. We define G as a diagonal matrix whose diagonal elements are the degrees of the vertices. Finally, we define our graph Laplacian as $L = G - W$. We can now use the smallest k eigenvalues of matrix L

Table 3: Results Spectral Clustering*

	$K = 2$		$K = 4$		$K = 8$		$K = 16$	
k	Size $\times 10^4$	Stats	Size $\times 10^4$	Stats	Size $\times 10^4$	Stats	Size $\times 10^4$	Stats
1	25.53	[129 194 237]	11.59	[129 164 193]	4.28	[100 114 127]	2.12	[116 122 138]
2	10.46	[20 90 128]	8.61	[64 99 129]	4.53	[128 142 156]	1.71	[72 79 86]
3			13.94	[194 220 237]	2.24	[41 58 70]	0.45	[30 39 44]
4			1.85	[20 50 63]	9.09	[217 228 237]	0.84	[45 51 57]
5					6.83	[186 200 216]	2.72	[172 178 184]
6					0.34	[20 34 40]	2.93	[198 204 211]
7					5.24	[157 172 185]	6.88	[227 230 237]
8					3.46	[71 85 99]	1.91	[87 101 94]
9							2.19	[143 157 150]
10							3.15	[212 220 226]]
11							2.14	[158 165 171]
12							3.19	[185 191 197]
13							0.06	[20 26 29]
14							2.33	[129 135 142]
15							2.04	[102 109 115]
16							1.33	[58 65 71]
Inertia	**6.83** $\times 10^7$		**2.12** $\times 10^7$		**3.1** $\times 10^6$		**2.3** $\times 10^6$	

**Stats* show the one-dimensional [min, mean, max] of a cluster. *Inertia* is the 1-D version of inertia defined in part A.

to define a vector for each observation in $\mathbb{R}^k$. Stacking these together forms a matrix $Z \in \mathbb{R}^{N \times k}$. Then K-means is applied to our transformed matrix Z to cluster the observations into k classes with the objective of minimising inertia, which is 1-D after the B&W transform. We stick to `sklearn`'s default settings, which means the K-means algorithm in `SpectralClustering()` is run with 10 different seeds. Eigenvalues are determined without an explicit solver strategy, which means we also have no eigendecomposition stopping criterion.

Note that when using spectral clustering we encounter the same problem as in the agglomerative hierarchical algorithm. Our image contains 360,000 pixels, meaning our similarity matrix will have a tremendous size and require around 520GB of internal memory. To overcome this problem we use the black and white transform again. Max pooling is impractical, as the maximum resolution that could be implemented in our case was 25x25, such that detail is almost non-existent (see figure 2a versus figure 2b). Even though we use a transformation, after the spectral clustering we can use an inverse transformation to recover the original image. Table 3 displays the results for spectral clustering using black and white filtering. As we display minimum and maximum brightness values in addition to means, we can very clearly see that the algorithm clusters on brightness. The maximum brightness of one cluster directly borders (within rounding margin) the minimum brightness of the next cluster for every value of K. Being able to more accurately seek these boundaries (increasing K) again monotonically decreases inertia, as expected.

Method comparison

A problem with comparing algorithms is that *inertia* in part A is not the same as in parts B/C, as the B&W transform makes inertia 1-D. We therefore have to consider inertias relatively, in which case we see that part C gives seemingly more sophisticated results than part B, although we are limited by our understanding of the domain if this is a meaningful clustering. As mentioned before, the spectral clustering uses K-means in the second stage of the algorithm, but on transformed data to improve efficiency. As the starting point is very important in determining the outcome of the clusters, a better starting point will result in lower cluster errors at the end. This could hint that part C outperforms part A, although we must remain careful because inertia is defined differently. Further, without domain knowledge, we explicitly refrain from commenting on the images produced by the methods. Based on the above arguments alone however, we may tentatively conclude that spectral clustering produces the 'best' results on our dataset.

References

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* http://www.deeplearningbook.org. MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Springer Series in Statistics. doi:10.1007/978-0-387-84858-7

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). *An introduction to statistical learning: With applications in r.* Springer Publishing Company, Incorporated.

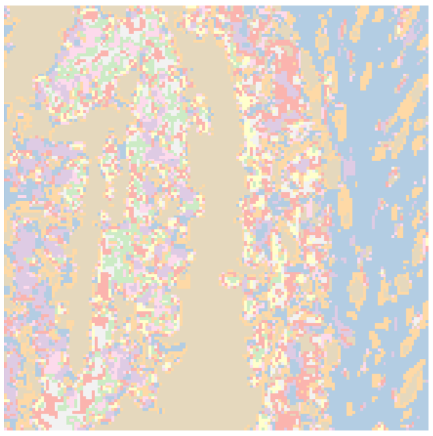
Johnson, R. A., & Wichern, D. W. (Eds.). (2002). *Applied multivariate statistical analysis.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Milligan, G. W. (1980). An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, *45*(3), 325–342. doi:10.1007/BF02293907

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems 14* (pp. 849–856). MIT Press. Retrieved from http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf

Figure 1a: Max pooling (150x150x3)

Figure 1b: Black and white transform



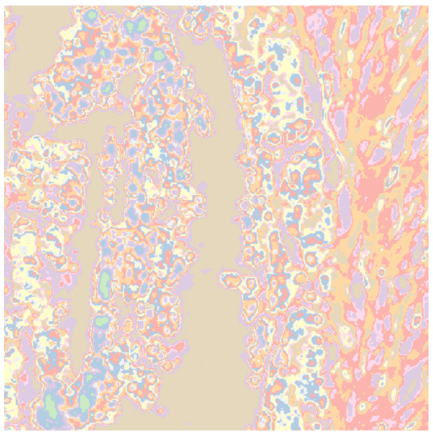
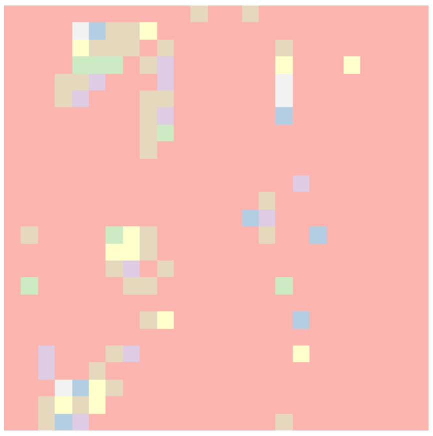


Figure 2a: Max pooling (25x25x3)

Figure 2b: Black and white transform



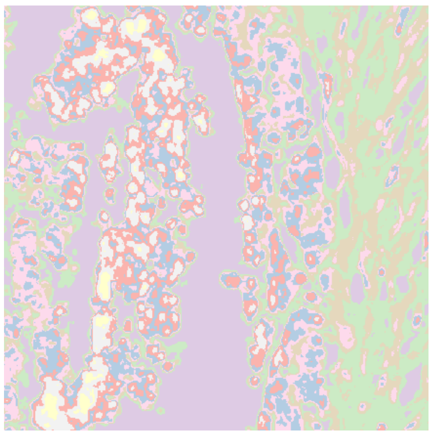


Figure 3a: KMeans - K = 2

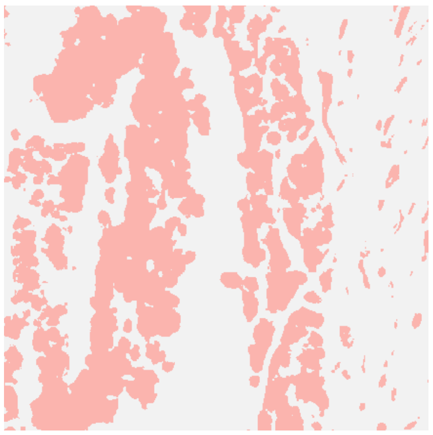


Figure 3b: KMeans - K = 4

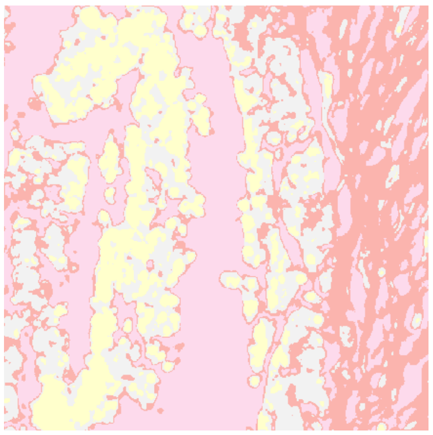


Figure 4a: KMeans - K = 8

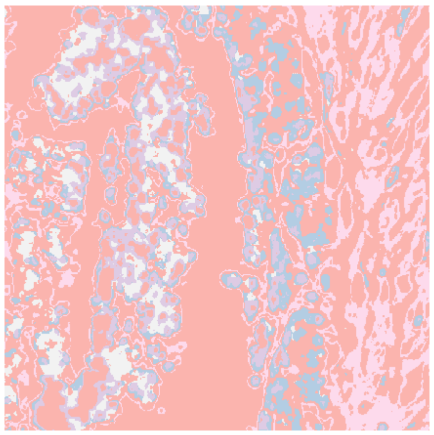


Figure 4b: KMeans - K = 16

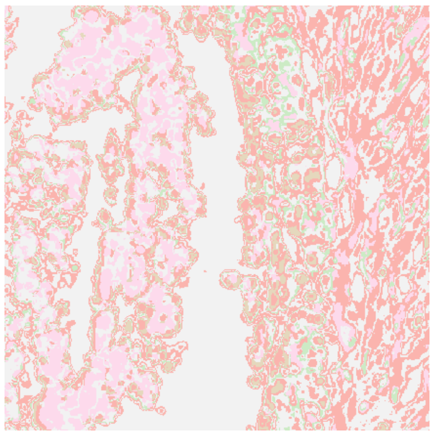


Figure 5a: Ward's linkage - K = 2

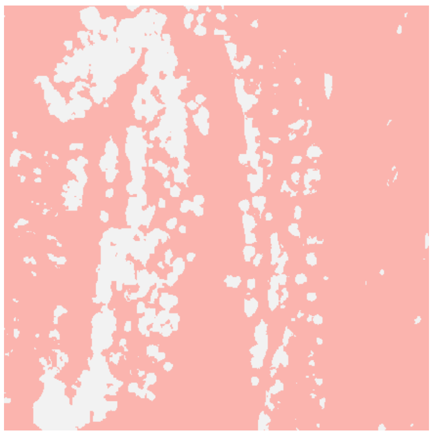


Figure 5b: Ward's linkage - K = 4

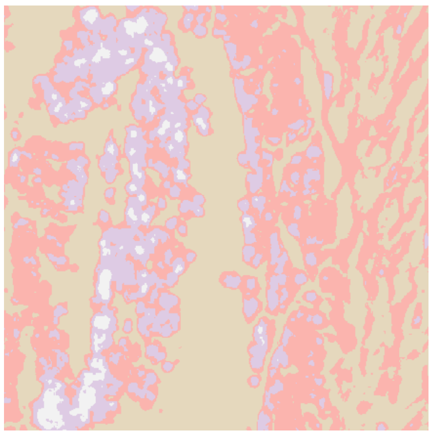


Figure 6a: Ward's linkage - K = 8

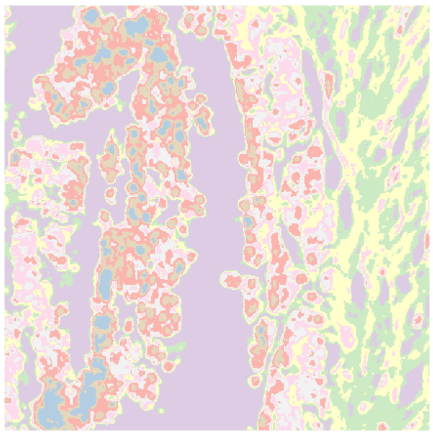


Figure 6b: Ward's linkage - K = 16

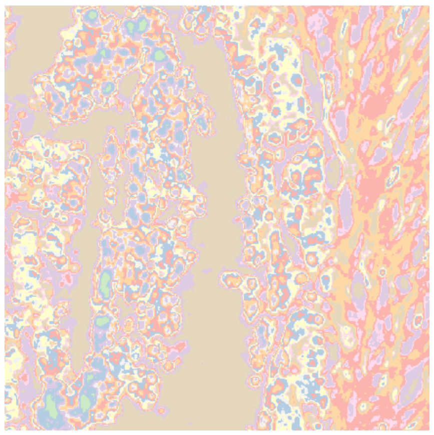


Figure 7a: Average linkage - K = 2

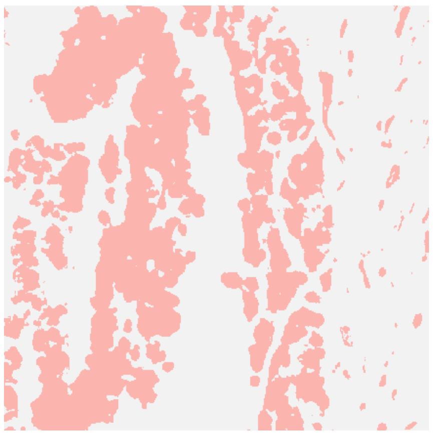


Figure 7b: Average linkage - K = 4

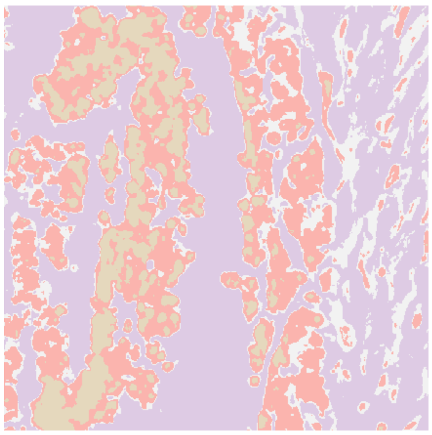


Figure 8a: Average linkage - K = 8

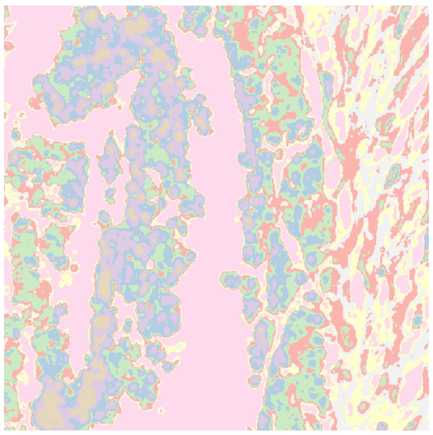


Figure 8b: Average linkage - K = 16

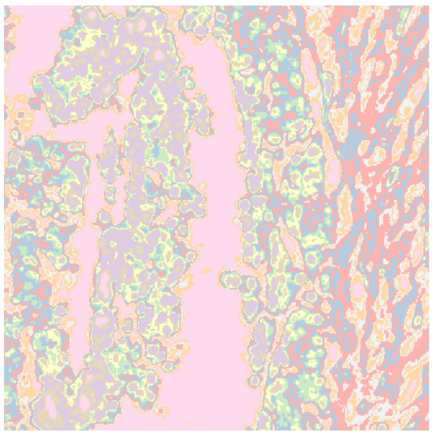


Figure 9a: Spectral clustering - K = 2

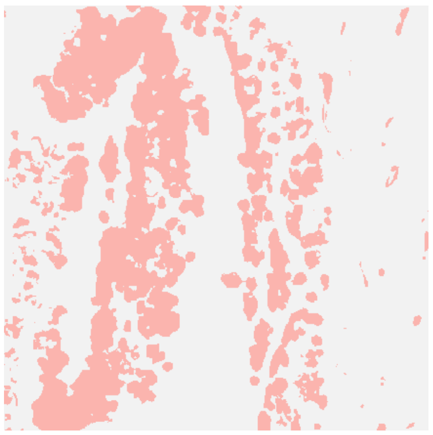


Figure 9b: Spectral clustering - K = 4

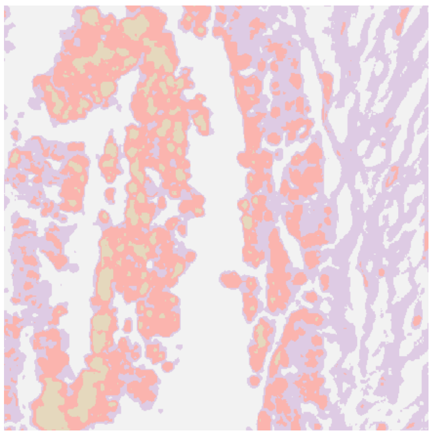


Figure 10a: Spectral clustering - K = 8

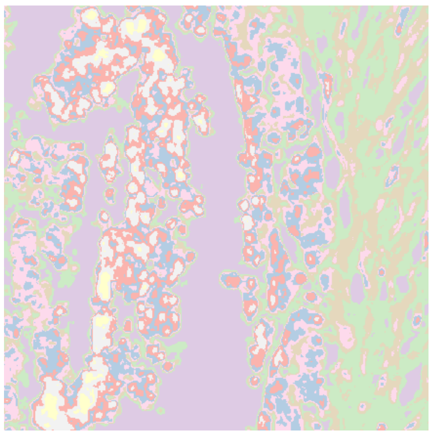
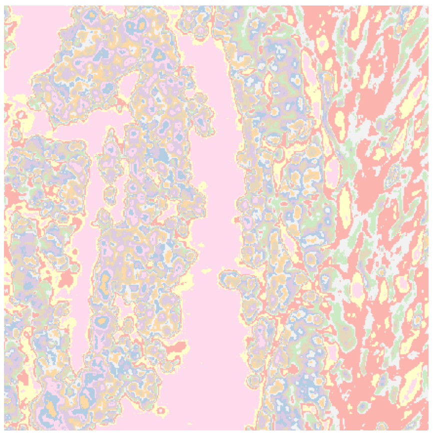


Figure 10b: Spectral clustering - K = 16



The notebook on the next page, and its images, can be accessed on:
https://drive.google.com/open?id=1wH4Vr23mz-4KxB-mThc2exQiNkjr2BCH

ML4

October 8, 2019

```
[ ]: import torch #PyTorch used for pooling operations, keep in mind installing this␣
      ↪library (non-CUDA) takes some time url: https://pytorch.org/get-started/
      ↪locally/
     import numpy as np
     import multiprocessing
     import collections
     import matplotlib.pyplot as plt
     from sklearn.cluster import KMeans, AgglomerativeClustering, SpectralClustering
     from PIL import Image
     from scipy.stats import describe

     # import sys
     # sys.path.append('/opt/tljh/user/lib/python3.6/site-packages/libsvm/')
     # import brisque
```

```
[ ]: #Set the number of jobs for each algorithm, its the number of available cores␣
      ↪minus two to keep the PC running at a decent pace
     n_jobs_count = multiprocessing.cpu_count()-2
```

0.1 User-defined functions

```
[ ]: def WarningSwitch(warning):
         import warnings
         warnings.filterwarnings(warning)
```

```
[ ]: def MaxPooling(image, size=(150,150,3)):

         resolution = image.shape[0]
         stride, kernel_size = [resolution//size[0] for _ in range(2)]

         pooler = torch.nn.MaxPool2d(kernel_size=kernel_size, stride=stride,␣
      ↪return_indices=True)
         image_as_tensor = torch.Tensor(np.expand_dims(image.T, axis=0))
         reshaped_tensor, inverse_indices = pooler(image_as_tensor)
         image_as_numpy = np.reshape(np.squeeze(reshaped_tensor.numpy().T), size).
      ↪astype(np.uint8)
```

```
    return image_as_numpy, inverse_indices
```

```
[ ]: def MaxUnpool(image, inverse_indices, original_size=(600,600,3)):

         resolution = image.shape[0]
         stride, kernel_size = [original_size[0]//resolution for _ in range(2)]

         unpooler = torch.nn.MaxUnpool2d(kernel_size=kernel_size, stride=stride)
         image_as_tensor = torch.Tensor(np.expand_dims(image.T, axis=0))
         unpooled_tensor = unpooler(image_as_tensor, indices=inverse_indices)

         image_as_numpy = np.reshape(np.squeeze(unpooled_tensor.numpy().T),␣
      ↪original_size).astype(np.uint8)

         return image_as_numpy
```

```
[ ]: def FinalUnpool(inverse_indices, labels, resized_shape=(150,150,3),␣
      ↪original_shape=(600,600,3)):

         resized_resolution = resized_shape[0]
         original_resolution = original_shape[0]
         stride = original_resolution//resized_resolution

         stride_blocks = np.
      ↪zeros((resized_resolution,resized_resolution,original_resolution**2//
      ↪(resized_resolution**2)), dtype=np.int32)
         pixels = np.zeros((original_resolution**2), dtype=np.int32)

         blocks = np.arange(0,original_resolution+stride,stride)

         for i in range(resized_resolution):
             for j in range(resized_resolution):
                 stride_blocks[i,j,:] = np.hstack(([np.
      ↪arange(blocks[j],blocks[j]+stride,1)+(blocks[i]+z)*original_resolution for z␣
      ↪in range(stride)]))

         indices = np.squeeze(inverse_indices.numpy()).T.
      ↪reshape(resized_resolution**2,resized_shape[2])

         for i in range(resized_shape[0]**2):
             row, column = np.where(stride_blocks==indices[i,0])[:2]
             pixels[stride_blocks[int(row), int(column)]] = labels[i]

         return pixels
```

```
[ ]: def ReverseBW(bw_image, unique_bw, shape, labels):

         bw_pixels = np.empty(shape)

         for i in range(unique_bw.shape[0]):
             indices = np.where(bw_image==unique_bw[i])
             bw_pixels[indices] = labels[i]

         image = np.dstack((bw_image, bw_pixels))

         return image
```

```
[ ]: def plot_image(image,plot_figsize, cmap, algorithm_name, cluster, path_images,␣
      ↪transform):

         plt.figure(figsize=plot_figsize)
         plt.imshow(image, cmap=cmap)

         plt.axis('off')
         plt.savefig(f"{path_images}{algorithm_name}_{cluster}_{transform}.svg",␣
      ↪bbox_inches=0)
```

```
[ ]: def ClusteringAlgorithm(image, algorithm, pooling=False, bw_transform=False,␣
      ↪pooling_shape=(150,150,3), plot=True, plot_figsize=(8,8), cmap='Pastel1',␣
      ↪algorithm_name='algoritm', cluster='4', path_images='./', warning='default'):

         WarningSwitch(warning)

         original_shape = image.shape

         if bw_transform:
             image_bw = np.array(Image.fromarray(image).convert("L"))
             unique_bw_image = np.expand_dims(np.unique(image_bw.reshape(image.
      ↪shape[0]**2)), axis=1)

             algorithm.fit(unique_bw_image)
             masked_image = ReverseBW(image_bw, unique_bw_image, image_bw.shape,␣
      ↪algorithm.labels_)

             if plot:
                 plot_image(masked_image[:,:,1], plot_figsize, cmap, algorithm_name,␣
      ↪cluster, path_images, "bw")

             #print(sorted(collections.Counter(masked_image[:,:,1].
      ↪reshape(original_shape[0]**2)).most_common()))
```

```
    else:
        if pooling:
            image, inverse_indices = MaxPooling(image, pooling_shape)
            algorithm.fit(image.reshape(image.shape[0]**2,image.shape[2]))

            image = FinalUnpool(inverse_indices, algorithm.labels_,␣
↪resized_shape=pooling_shape, original_shape=original_shape)

            if plot:
                image_for_plot = image.
↪reshape(original_shape[0],original_shape[1]).T
                plot_image(image_for_plot, plot_figsize, cmap, algorithm_name,␣
↪cluster, path_images, "pooling")

        else:
            algorithm.fit(image.reshape(image.shape[0]**2,image.shape[2]))

            if plot:
                image_for_plot = image[:,:,0].
↪reshape(original_shape[0]**2)[algorithm.labels_].
↪reshape(original_shape[0],original_shape[1])
                plot_image(image_for_plot, plot_figsize, cmap, algorithm_name,␣
↪cluster, path_images, "none")

        print(sorted(collections.Counter(algorithm.labels_).most_common()))

    if bw_transform:
        return_image = masked_image
    else:
        return_image = image

    return return_image
```

0.2 Load image & define clusters

[ ]:
```
image = np.array(Image.open("sample.jpg"))

clusters = [2,4,8,16]
```

0.3 KMeans Clustering

```
[ ]: %%time

     #If the "worker stopped" warning shows up, lower the n_jobs

     for cluster in clusters:

         kmeans = KMeans(n_clusters=cluster, n_jobs=n_jobs_count)

         ClusteringAlgorithm(image, kmeans, pooling=False, bw_transform=False,␣
      ↪algorithm_name="KMeans", cluster=cluster, plot=True, path_images='./images/
      ↪', warning='ignore')
```

0.4 Hierarchical Agglomerative Clustering

```
[ ]: %%time

     clustered_images = np.empty((2,4,600,600,2))

     file = open("agglomerative_results.txt","w+")

     #B&W transformation

     linkage_methods = ['ward', 'average']

     for i, linkage in enumerate(linkage_methods):

         for j, cluster in enumerate(clusters):

             AGNES = AgglomerativeClustering(n_clusters=cluster, linkage=linkage)

             clustered_images[i,j,:,:] = ClusteringAlgorithm(image, AGNES,␣
      ↪pooling=False, bw_transform=True, pooling_shape=(150,150,3),␣
      ↪algorithm_name=f"AGNES_{linkage}", cluster=cluster, plot=True, path_images='.
      ↪/images/')

         linkage_string = f"Linkage method: {linkage}"
         print(linkage_string)
         file.write(f"{linkage_string} \n")



         for index, cluster in enumerate(clusters):
```

```
        inertia = 0

        cluster_string = f"Cluster: {cluster}"
        print(cluster_string)
        file.write(f"{cluster_string} \n")
        for label in range(cluster):

            pixels, minmax, mean, *_ = describe(clustered_images[i,index,:,:
 ↪,0][clustered_images[i,index,:,:,1]==label])
            descriptive_string = f'{label} - pixels: {pixels} \t minmax:␣
 ↪{minmax} mean: {mean:.2f}'

            inertia = inertia + np.sum((clustered_images[i,index,:,:
 ↪,0][clustered_images[i,index,:,:,1]==label]-mean)**2)

            print(descriptive_string)
            file.write(f"{descriptive_string} \n")

        inertia_string = f"Inertia: {inertia:.4f}"
        print(inertia_string)
        file.write(f"{inertia_string} \n")

file.close()
```

[ ]:
```
%%time

#Pooling transformation

linkage_methods = ['ward', 'average']

for linkage in linkage_methods:

    for cluster in clusters:

        AGNES = AgglomerativeClustering(n_clusters=cluster, linkage=linkage)

        ClusteringAlgorithm(image, AGNES, pooling=True, bw_transform=False,␣
 ↪pooling_shape=(150,150,3), algorithm_name=f"AGNES_{linkage}",␣
 ↪cluster=cluster, plot=True, path_images='./images/')
```

0.5 Spectral Clustering

```
[ ]: %%time

     clustered_images = np.empty((1,4,600,600,2))

     file = open("spectral_results.txt","w+")

     #If pooling, set assign_labels='discretize'

     for i, cluster in enumerate(clusters):

         spectral = SpectralClustering(n_jobs=n_jobs_count, n_clusters=cluster)

         clustered_images[0,i,:,:,:] = ClusteringAlgorithm(image, spectral,␣
      ↪pooling=False, bw_transform=True, pooling_shape=(25,25,3),␣
      ↪algorithm_name=f"Spectral", cluster=cluster, plot=True, path_images='./
      ↪images/')


     for index, cluster in enumerate(clusters):

         cluster_string = f"Cluster: {cluster}"
         print(cluster_string)
         file.write(f"{cluster_string} \n")

         inertia = 0

         for label in range(cluster):

             pixels, minmax, mean, *_ = describe(clustered_images[0,index,:,:
      ↪,0][clustered_images[0,index,:,:,1]==label])
             descriptive_string = f'{label} - pixels: {pixels} \t minmax: {minmax}␣
      ↪mean: {mean:.2f}'
             print(descriptive_string)
             file.write(f"{descriptive_string} \n")

             inertia =+ np.sum((clustered_images[0,index,:,:
      ↪,0][clustered_images[0,index,:,:,1]==label]-mean)**2)

         inertia_string = f"Inertia: {inertia:.4f}"
         print(inertia_string)
         file.write(f"{inertia_string} \n")

     file.close()
```