DD2423 Image Analysis and Computer Vision

Lab 3: Image Segmentation

Dario Del Gaizo, Giulia Rebay, Isabella Rositi

16 December, 2022

Question 1: How did you initialize the clustering process and why do you believe this was a good method of doing it?

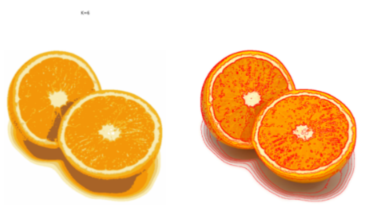
The random initialization avoids bias and is computationally efficient. Some methods choose the initial centroids to be the ”representative” points of the dataset, such as the points with the highest variance or the points that are furthest from each other (in this case white and orange). These methods can help to improve the convergence of the algorithm and to produce more balanced clusters. In this case we could also apply the split of pixels into ordered K subsets and then pick the centers randomly from the subsets, this takes centers that are spread out more.

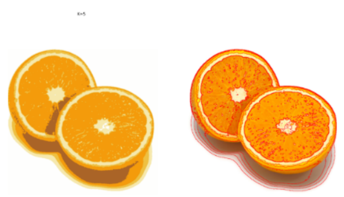
Question 2: How many iterations L do you typically need to reach convergence, that is the point where no additional iterations will affect the end results?

It depends on the conversion delta, the initial centers, the complexity of the image (diversity in pixels translates in more computational time) and the applied blurring. In this case when the delta reaches below 0.01 as a change in the mean, it stops. The number of iterations increases as the number of clusters increases.

Question 3: What is the minimum value for K that you can use and still get no superpixel that covers parts from both halves of the orange? Illustrate with a figure.

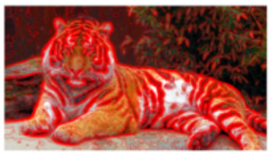
$K = 6$ divides the two halves of the orange while $K = 5$ doesn't.





Question 4: What needs to be changed in the parameters to get suitable superpixels for the tiger images as well?

First, increasing the number of clusters. As the pixels vary a lot, having more centers will spread them more in different clusters. The number of iterations has to be higher in order to converge. Alternatively, the blurring can be increased to reduce the pixel variance.



Question 5: How do the results change depending on the bandwidths? What settings did you prefer for the different images? Illustrate with an example image with the parameter that you think are suitable for that image.

Spatial bandwidth: how large is the mean calculation? Increasing the spatial bandwidth means taking more pixels in the mean calculation. The effect is a decreasing in the general variance of pixels as the uni-color regions will be larger, so less modes. If s_{bw} is small, then we have more modes and more accurate pixel assignment. If s_{bw} is large, then we have a broader Gaussian that results in blended neighbours.

Colour bandwidth: the radius of the colour space, the higher it is, the more non-similar colours will be considered the same.

These are two similar concepts and work similarly, but taking into account different properties such as position and colour.

Question 6: What kind of similarities and differences do you see between K-means and mean-shift segmentation?

Similarities: Both K-means and mean-shift are clustering algorithms, which means that they are used to group similar data points together into clusters, in this case about image segmentation. Both algorithms use iterative procedures to find clusters.

Differences: Mean-shift considers spatial information. K-means needs a number of clusters and mean-shift needs bandwidths. Outliers are more of a problem in K-means compared to mean-shift. In mean-shift you perform gradient-ascent to find the modes to each pixel, in K-means you find the clusters for all pixels simultaneously.

Question 7: Does the ideal parameter setting vary depending on the images? If you look at the images, can you see a reason why the ideal settings might differ? Illustrate with an example image with the parameters you prefer for that image.

Optimal parameter settings greatly depend on image complexity and general feature distribution. Specifically, each parameter in the *apply_normCuts* function affects the way the cutting of the image is done in a unique and different way:

- *ncuts_tresh*: impacts the maximum cutting cost that we allow. The higher its value the more we will be able to separate apart similar areas
- *min_area*: sets the lower size threshold on segments or image areas that can be separated apart
- *max_depth*: sets the maximum amount of cuts which are allowed
- *colour_bandwidth*: affects the weight given to similar pixels (the lower the bandwidth, the more we will be able to differentiate between similar and not so similar areas in the image)
- *radius*: controls the dimension of the neighborhood we consider pixel per pixel

By reflecting on the effect each parameter has on the Ncuts algorithm, we can easily understand that the optimal parameter combination will greatly depend on the image at hand. Indeed, when segmenting more complex and articulate images, we will have to pick a set of parameters which place greater importance on better differentiating areas. This will usually imply higher values of *ncuts_tresh*, lower values of *colour_bandwidth* and lower values of *min_area.*

Therefore, by looking at the four images (tiger1, tiger2, tiger3 and orange), we will expect higher values of *colour_bandwidth* and *min_area* and lower values of *ncuts_tresh* for images with less complex features and details. Moreover, we will also expect so when there are less colors appearing in the image (thereby differentiating between areas). This will be the case, for example, for the "orange" image, whereas opposite conclusions will be drawn when segmenting "tiger3" which is the most complex and detailed picture out of them all.

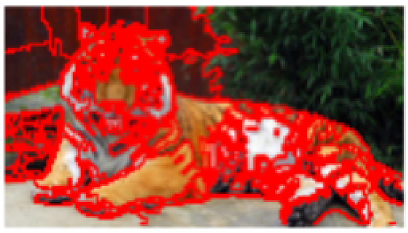
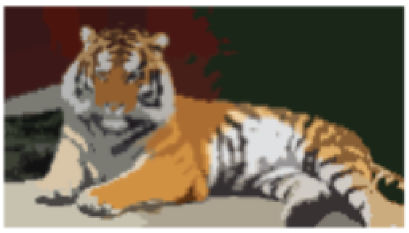
In brief:

- The higher the complexity of the image, the more *ncuts_tresh* will have to be increased to achieve effective segmentation
- The higher the complexity of the image, the higher *max_depth* will have to be, in order to allow for detailed segmentation
- The higher the *radius*, the more we will be able to group together far away pixels. This will usually result in less segments.

We have here included some of the best parameter combinations for the four images at hand.

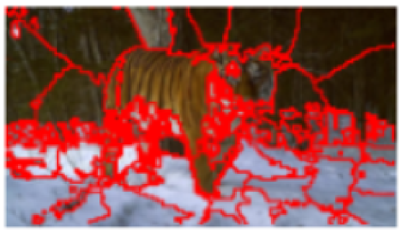
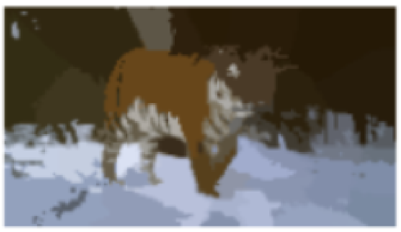
tiger1:

apply_normCuts(tiger1, *colour_bandwidth* = 15, *radius*= 10, *ncuts_tresh* = 0.5, *min_area* = 20, *max_depth* = 6)

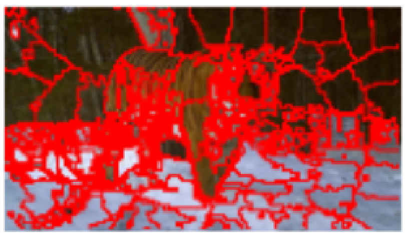
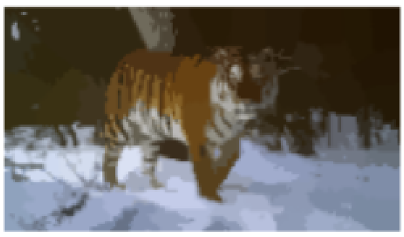



tiger2:

apply_normCuts(tiger2, *colour_bandwidth* = 25, *radius*= 10, *ncuts_tresh* = 0.8, *min_area* = 20, *max_depth* = 12)

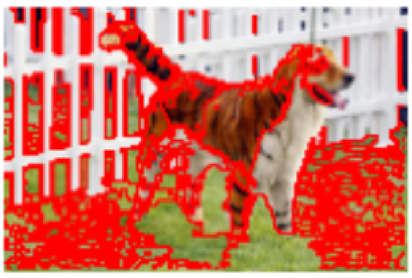
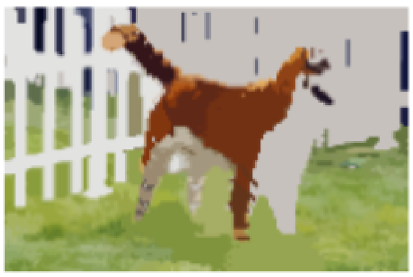



apply_normCuts(tiger2, *colour_bandwidth* = 25, *radius*= 10, *ncuts_tresh* = 0.5, *min_area* = 20, *max_depth* = 12)

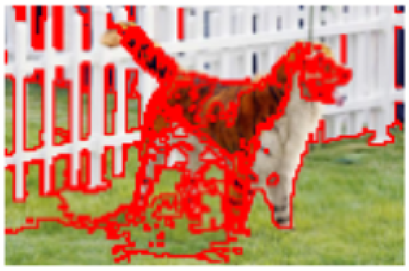
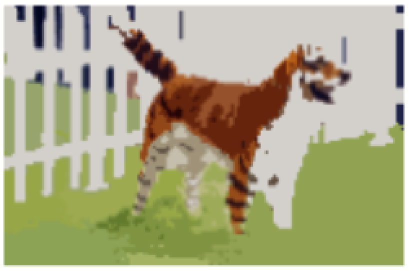



This comparison allows us to see how changing *ncuts_tresh* affects the final result: less segmentation for lower values of *ncuts_tresh* (from 0.8 to 0.5) tiger3:

apply_normCuts(tiger3, *colour_bandwidth* = 10, *radius*= 10, *ncuts_tresh* = 0.5, *min_area* = 20, *max_depth* = 10)

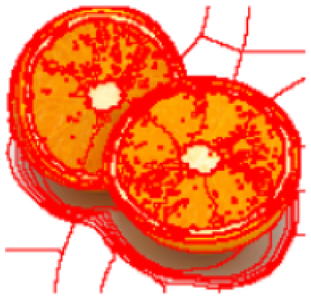
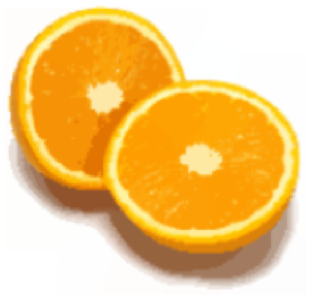



apply_normCuts(tiger3, *colour_bandwidth* = 10, *radius*= 10, *ncuts_tresh* = 0.5, *min_area* = 10, *max_depth* = 10)

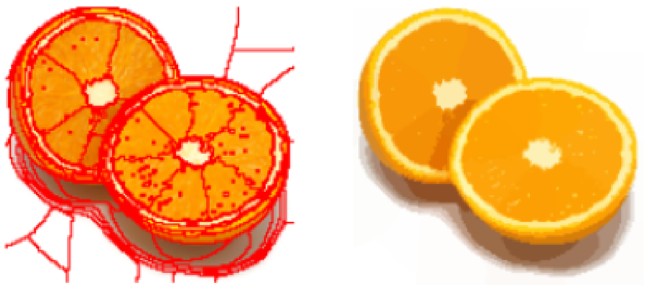



orange:

apply_normCuts(orange, *colour_bandwidth* = 10, *radius*= 15, *ncuts_tresh* = 0.5, *min_area* = 60, *max_depth* = 10)

apply_normCuts(orange, *colour_bandwidth* = 20, *radius*= 15, *ncuts_tresh* = 0.5, *min_area* = 60, *max_depth* = 10)



Question 8: Which parameter(s) was most effective for reducing the subdivision and still result in a satisfactory segmentation?

The most effective parameters that allowed us to achieve a satisfactory segmentation were: *ncuts_tresh min_area max_depth colour_bandwidth* for all the reasons already explained in the section above.

Question 9: Why does Normalized Cut prefer cuts of approximately equal size? Does this happen in practice?

This happens due to the mathematical formulation of the problem. Indeed, Normalized Cut aims to find the partition of the vertex space V into two subsets A and B such that

$$N_{cut}(A, B) = cut(A, B)assoc(A, V) + cut(A, B)assoc(B, V)$$

is minimized.

The quantity $assoc(S, V)$, where S is a generic subset of V, counts all possible edges connected to any vertex belonging to S. Therefore:

$$assoc(V) = assoc(A, V) + assoc(B, V) - cut(A, B)$$

where $cut(A, B)$ is removed in order to avoid counting the connecting edges twice. This means we can express quantity $assoc(B, V)$, appearing at the second denominator of the $N_{cut}(A, B)$ formula above, as

$$assoc(B, V) = assoc(V) - assoc(A, V) + cut(A, B)$$

By rewriting the $N_{cut}(A, B)$ equation with the quantity above and differentiating with respect to $assoc(A, V)$, we can minimize $N_{cut}(A, B)$ by setting the derivative to zero. We will here skip extensive computations, but this will lead us to the following result:

$$assoc(A, V) = assoc(B, V)$$

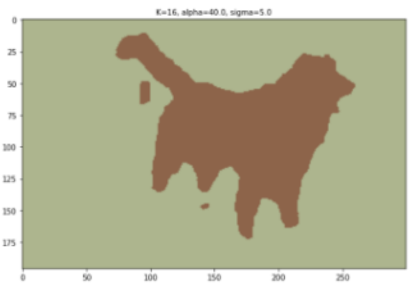
which explains why the optimal result is achieved when cuts are of equal size. In practice, however, due to all the other existing parameters which are defined in the function call, the final proportionality of vertices in each subset will differ from the ideal case.

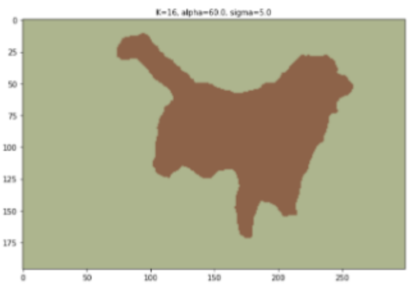
Question 10: Did you manage to increase *radius* and how did it affect the results?

By increasing *radius* we allowed for bigger and more widespread neighborhoods to be included in computations. This greatly impacted CPU time, resulting in a greater cost and in slower results. When increasing the radius, we achieved larger segments and areas, but suffered in detailed color differentiation, since more pixels (of possibly slightly different colors) were clumped together.

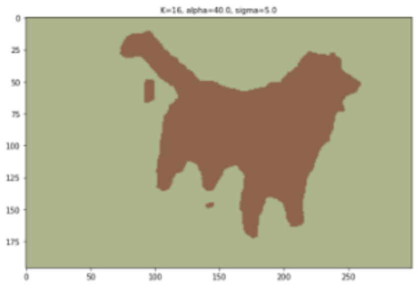
Question 11: Does the ideal choice of alpha and sigma vary a lot between different images? Illustrate with an example image with the parameters you prefer.

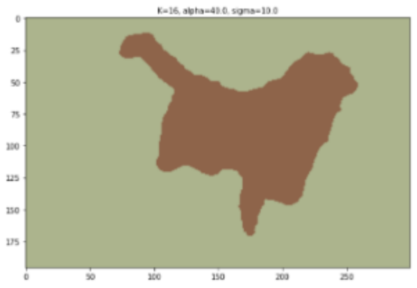
alpha is the maximum cost that an edge can have. As alpha increases, we assign the edges a higher maximum cost, meaning that small details that are "similar" to the foreground will be cut out.





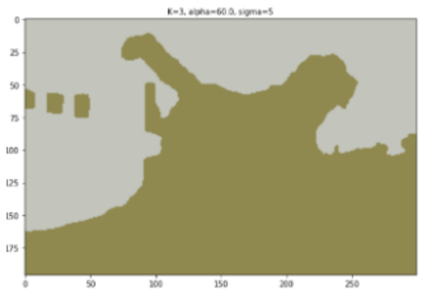
sigma is how much the edge cost decays for decreasing similarity between neighboring pixels. As sigma increases, the more pixels that have lower similarity will be penalized. If we have higher values of sigma, we will have less detailed separation lines.

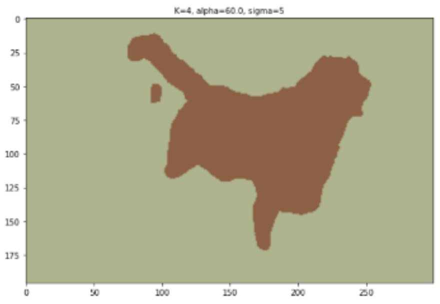


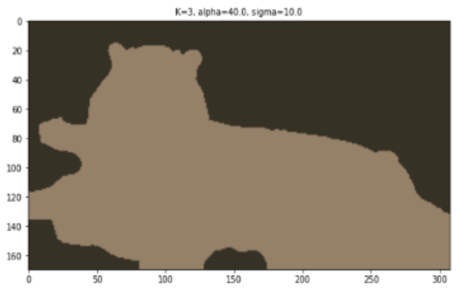


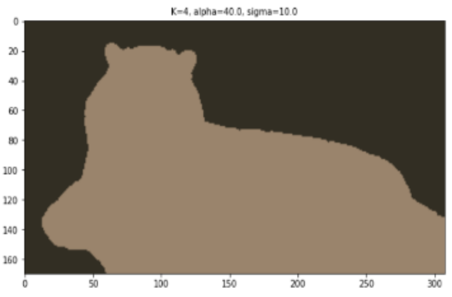
However, the values of both parameters vary based on the details of the image. If we have very detailed images, lower alphas could pick up too much detail from the background and treat it as foreground (or vice versa). However, we can achieve similar results by either increasing alpha or sigma.

Question 12: How much can you lower K until the results get considerably worse?





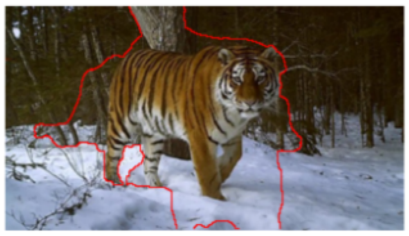


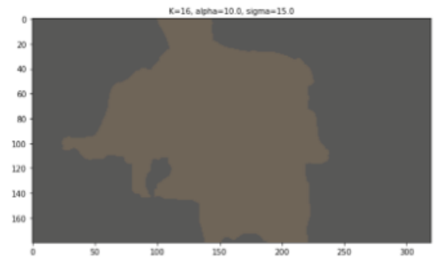


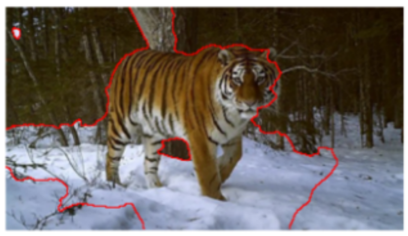
We can see that for most images, with $K = 4$ we still have acceptable subdivisions in foreground and background. For K=3, however, we see that the major figures are not identified anymore, meaning that applying only 3 Gaussian Kernels is not enough to assign satisfactory probability colors to the pixels. This means that the statistical models that rule the probability of a pixel belonging either to foreground or background are not satisfactory when implementing only 3 gaussian kernels.

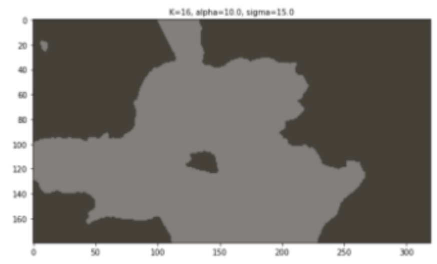
Question 13: Unlike the earlier method Graph Cut segmentation relies on some input from a user for defining a rectangle. Is the benefit you get of this worth the effort? Motivate!

Providing a correct area is extremely important in order to be able to successfully implement the GraphCut method. Indeed, the wrong area can compromise the results of the segmentation, since the coordinates define the area of the image containing the foreground. Indeed, even if changed slightly, the method can provide different results: it focuses on different parts of the image when it performs segmentation, as shown below. The first representation focuses more on the central area of the image vertically, and the second representation horizontally.









Moreover, if the areas provided are too wide (like in these cases), the method will include in the foreground also elements of the background. Finding the perfect area can be extremely time consuming and computationally costly, however it can improve the results of the segmentation drastically.

Question 14: What are the key differences and similarities between the segmentation methods (K-means, Mean-shift, Normalized Cut and energy-based segmentation with Graph Cuts) in this lab?

K-means is a clustering algorithm that partitions an image into a predetermined number of clusters, based on the color and intensity of the pixels in the image. It is a simple and efficient method, but it can be sensitive to the initial conditions and to outliers.

Mean-shift, on the other hand, is a non-parametric algorithm (user still needs to specify the spatial and colour bandwidth) that does not require the user to specify the number of clusters in advance. Instead, it uses a sliding window to iteratively estimate the mode (i.e. the most dense location) of the pixel intensities in the image, and assigns each pixel to the cluster associated with its mode. This makes Mean-shift more flexible and robust than K-means, but it can be computationally expensive (and it shows in the results).

Normalized Cut is another popular method for image segmentation. It is based on the idea of minimizing the total "cut" between the pixels in an image, while simultaneously maximizing the similarity of the pixels within each cluster. This method is effective at preserving the boundaries between different objects in an image, but it can be sensitive to noise and may not always produce the best results.

Energy-based segmentation with Graph Cuts is a more advanced method that uses graph theory to represent the relationships between the pixels in an image. It involves defining an "energy" function that measures the likelihood of a given segmentation, and then using optimization algorithms to find the segmentation that minimizes this energy. This method is more powerful and flexible than the other methods mentioned above, but it can be computationally expensive since it takes extensive parameter tuning and may require a large amount of training data to produce good results.

Overall, the main similarity between these methods is that they all aim to partition an image into distinct groups or regions, based on the color and intensity of the pixels in the image. The key differences between these methods lie in the specific algorithms and techniques they use to achieve this goal. The first two methods look at neighbourhoods and take into account either only colour (K-means) or both colour and spatial information (Mean-shift) in order to cluster the pixels. The last two methods are graph-based: they represent the whole image as a graph and then proceed to cut the edges between nodes (the pixels) that have higher cost in order to create regions of pixels that are similar to each other.