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

If you know the colors of the image and how the image looks like, One can initialize the cluster centers in the according color spectrums. The initial distance between pixels and cluster center will then be smaller and less iterations/better results can be achieved.

Seed
In the general case then the initial cluster centers can be randomly generated using the seed. We can test different seed values to make sure we don't have a worst case/best case scenario.

Number of clusters
We use a smaller value of clusters K in range 2-6 because the picture of the oranges is mostly just 2 colors so we don't need that many clusters to get a good segmentation.

Variance:
The orange image is a simple image with just 2 main colors. We don't need a detailed image to separate the pixels into clusters in this case. Therefore, we can smooth the image with a larger image_sigma to get rid of unnecessary details. Another observation was that iterations needed to converge lowered when we smoothed the image more.

Scaling:
Increasing image size and resolution by a scale factor is time consuming but increases details. Decreasing means losing information.

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?

- The number iterations needed depends on how we initialize the cluster centers i.e. which seed we used etc.
- Iterations depend on variance used when smoothing, $variance \Uparrow \ iterations \Downarrow$
- Iterations depend on scale_factor used
- Most significant dependency is with number of clusters $K \Uparrow \ iterations \Uparrow$

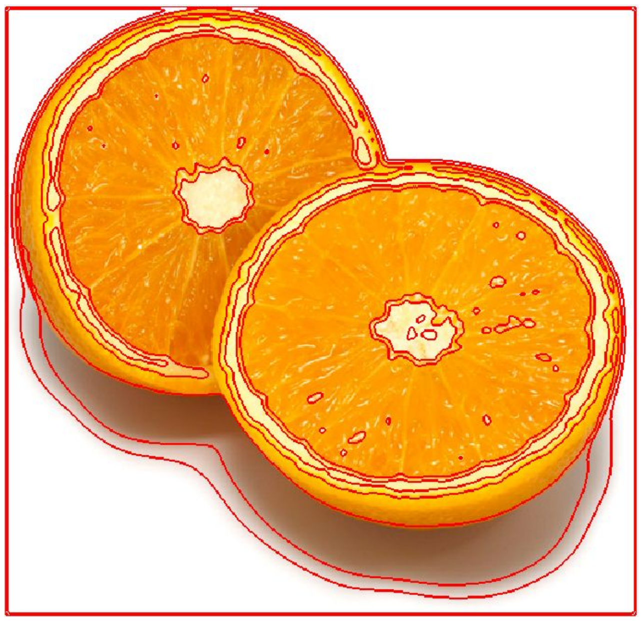
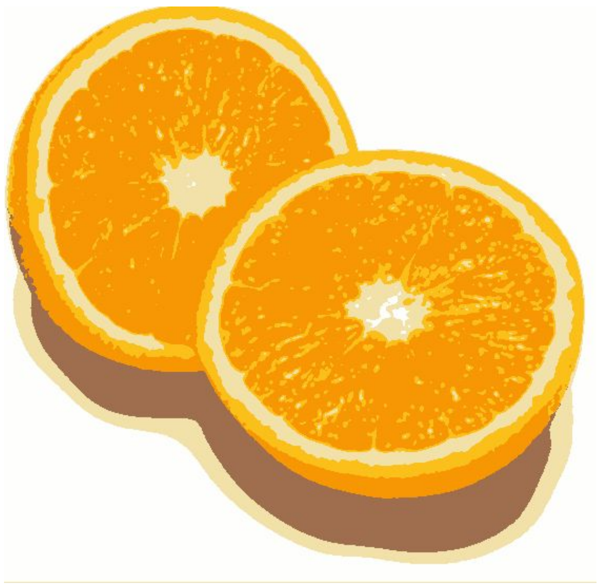
We can write the code so that we stop iterating and updating the centers of the clusters when the difference between each iteration is below a certain threshold value. We set the threshold to 0.01 and noted the iterations for each value of tested K.

Convergence: Variance = 1. Scalefactor = 1, seed = 14				
K	Orange	Tiger1	Tiger2	Tiger3
3	11	26	59	7
5	32	82	49	47
10	89	68	106	58

As we can see, the iterations can sometimes decrease when K increases marked in red. That is because of randomize process of the added cluster centers.

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.

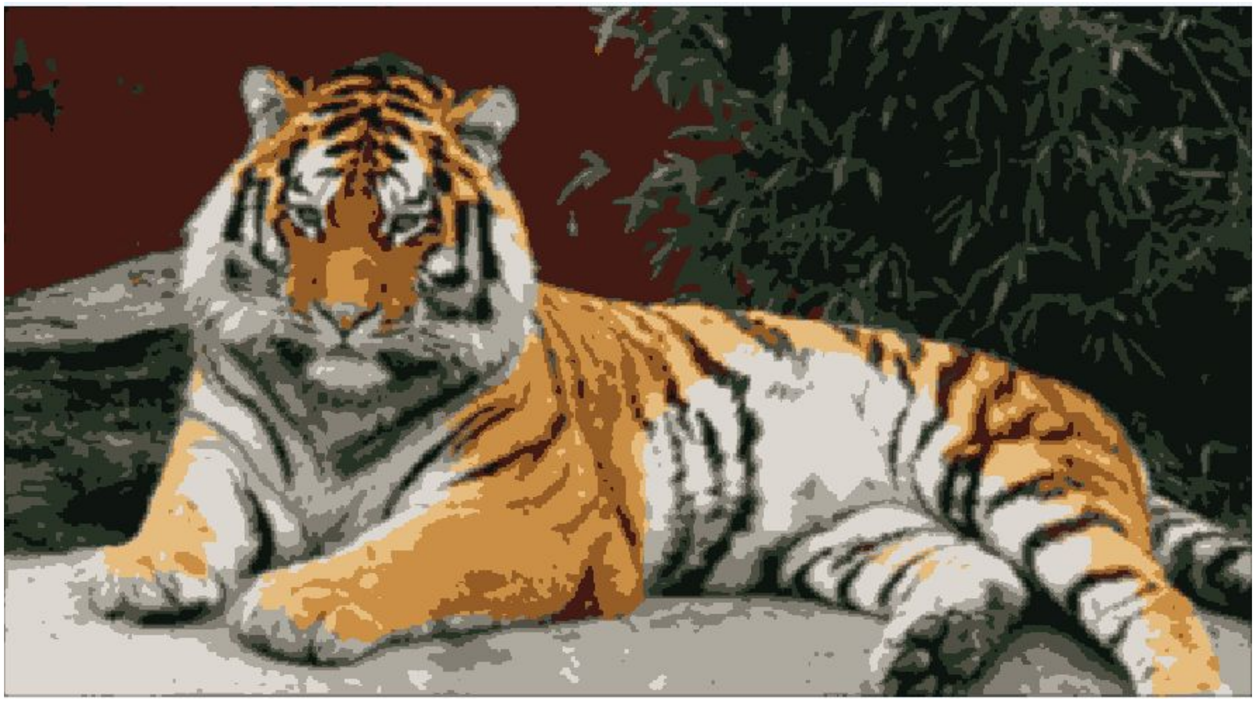
K = 5, Variance = 1, L = 15, Scalefactor = 1

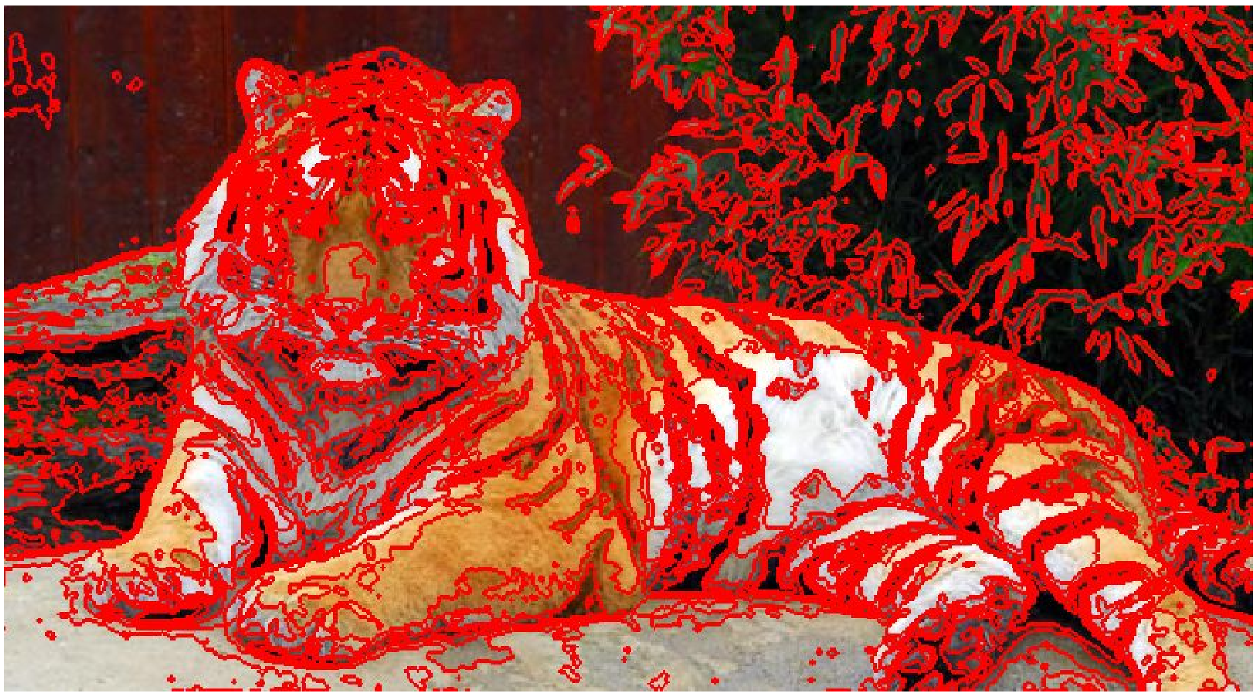



Question 4: What needs to be changed in the parameters to get suitable superpixels for the tiger images as well?

The other images are more complex than the orange image, they have more colors present and more detail. This forces us to increase number of clusters K to get a good result. As seen in question 2 we need to increase L to reach convergence when K increases.

K = 10 , Variance = 1, L = 30 , Scalefactor = 1





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

Goal: As large segments as possible but not covering more than one object

Observations:

- The values of the $color\_bandwidth = O_c$ and the $spatial\_bandwidth = O_s$ tell us how much the pixel representation in the feature space will be weighted on spatial position and color of the pixel. If the O_c value is higher than O_s value we put more significance on color of the image etc.

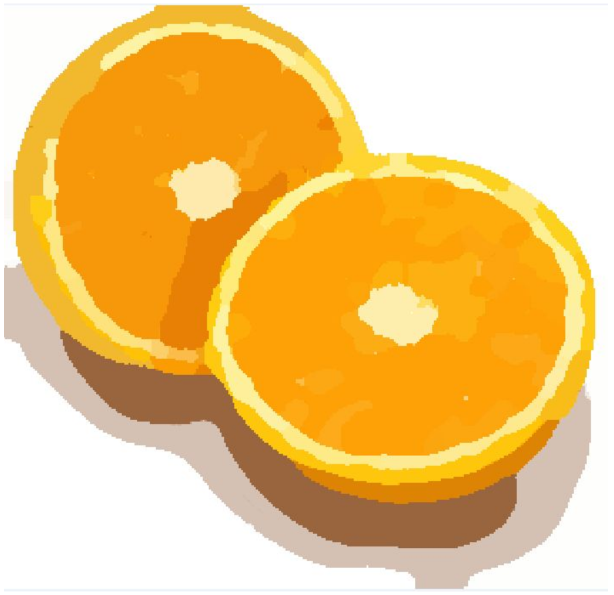
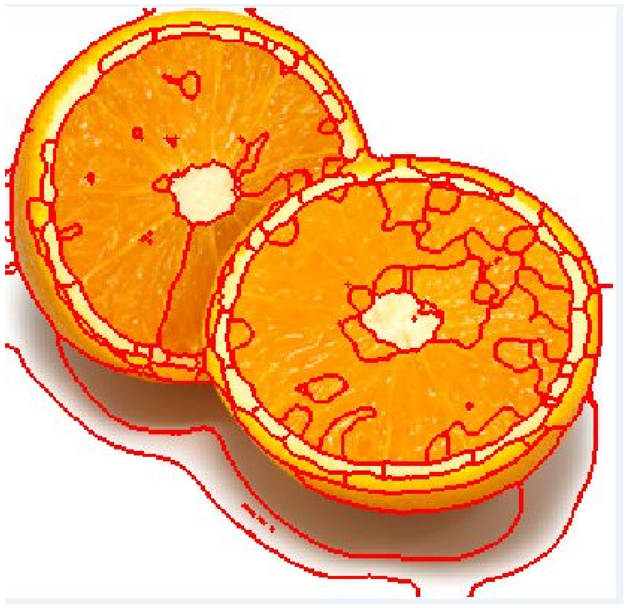
- Gaussian smoothing the image beforehand decreases difference between pixel color, larger variance will therefore give positions closer to each other the feature space, we will therefore see fewer attraction basins leading to fewer modes.

- Decreasing O_s produces more small and compact attraction basins leading to more modes. Region of interest becomes smaller and the gaussian kernel converges more locally

- Decreasing O_c gives more segments in the produced image

- Increasing O_s increases computational time

$O_s = 4$, $O_c = 6$

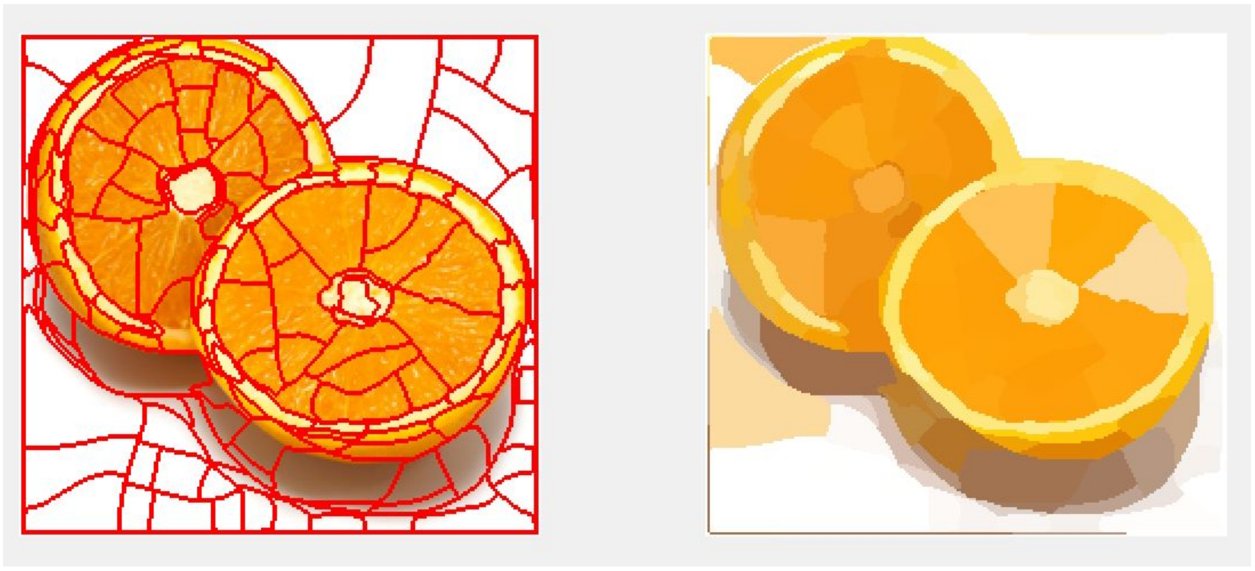
Question 6: What kind of similarities and differences do you see between K-means and mean-shift segmentation?

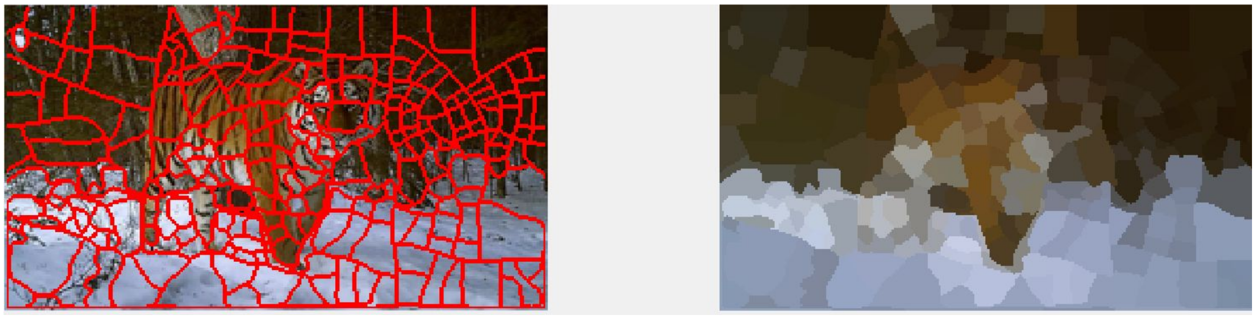
- K-means requires us to define number of clusters beforehand while mean-shift segmentation produces an unknown number of modes which depends on the window size i.e. region of interest

- K-means does not take into account position of the pixels, the location in the xy-plane, Therefore, we can not separate between two different objects that are grouped in the same cluster.
  In our case the K-means would place all pixels of the same color in the same group of superpixels whereas mean shift can distinguish objects with the same color because we work in the 5-dimensional space, taking into account the position and color of pixels.

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 dier? Illustrate with an example image using the parameters you prefer for that image.

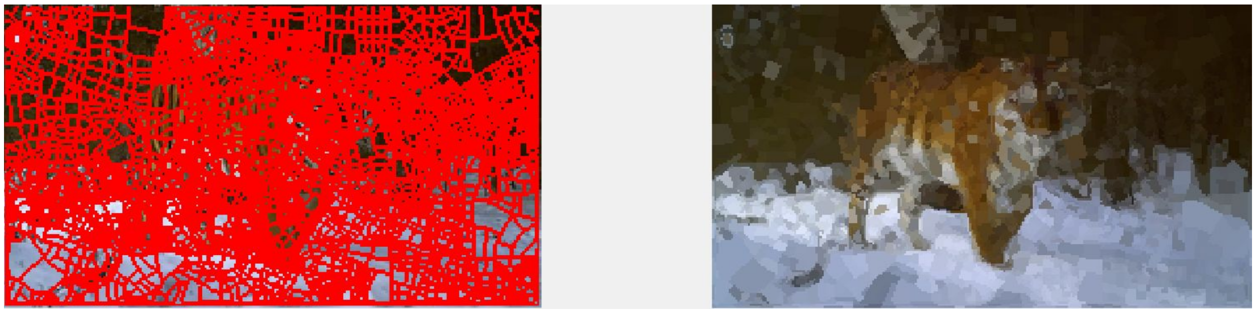
Ideal Parameters for Orange:
colour_bandwidth = 20.0;
radius = 3;
ncuts_thresh = 0.4;
min_area = 50;
max_depth = 8;
scale_factor = 0.4;
image_sigma = 2.0;





We ran the same parmeters for two images Orange and Tiger2. The observation is clear that the ideal paramaters change depending on the image.

Ideal parameters for Tiger 2:
colour_bandwidth = 20.0;
radius = 3;
ncuts_thresh = 5;
min_area = 5;
max_depth = 12;
scale_factor = 0.4;
image_sigma = 2.0;



The tiger image is much more detailed. To get a better result we

- Reduced minimum area to catch the futures of smaller details in the image
- Increased max_depth to create more cuts
- Increased Threshold to allow for more cuts in colors that are similar to each other, i.e. where the cost to cut is higher. This way we produce a more detailed result

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

Threshold, min_area, and max_depth proved to be effective for reducing subdivision.
We need to find optimal setting for min_area.threshold and max_depth to optimize results while limiting subdivision.

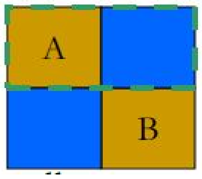
Question 9: Why does Normalized Cut prefer cuts of approximately equal size? Does this happen in practice?

The method works such that we want to minimize $Ncut(A, B)$ by splitting the image into two parts. We need to find the ideal cut. The Normalized cut normalizes for volume of segments.

$$Ncut(A, B) = \frac{Cut(A,B)}{assoc(A,V)} + \frac{Cut(A,B)}{assoc(B,V)}$$

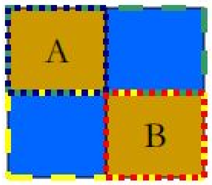
$assoc(A, V)$ *is the sum of the cost of edges that touch* A , $assoc(B, V)$ *is the total cost of all edges that touch* B. Also called the volumes of the sets A and B.

$$assoc(A, V)$$



The Normalized Associations can be written as

$$Nassoc(A, B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)}$$



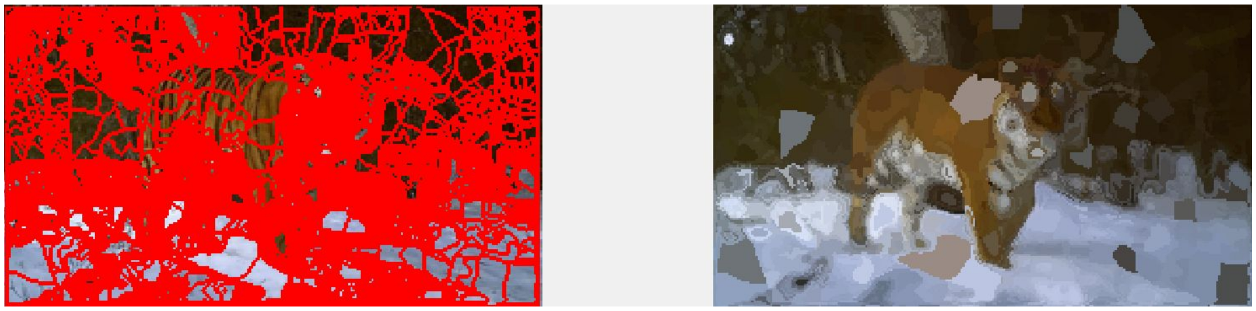
Reflects how tightly on average nodes within the group are connected to each other.

Another important property of association and disassociation of a partition is that they are naturally related. We can rewrite as

$$Ncut(A, B) = 2 - Nassoc(A, B)$$

We then see the criterion for minimizing Ncut between groups is equal to Maximizing association within groups. Which gives a bias to equal size partitions. Minimizing the disassociation between the groups and maximizing the association within the group are identical.

Question 10: Did you manage to increase radius and how did it affect the results?
Try to increase the radius to include neighbouring pixels that are a bit further away from each other. This usually leads to a better segmentation, but at the cost of slower computations.



Keeping all other parameters the same we increased the radius to 10

colour_bandwidth = 20.0;
radius = 10;
ncuts_thresh = 5;
min_area = 5;
max_depth = 12;
scale_factor = 0.4;
image_sigma = 2.0;

One of the goals of segmentation is to have as large segments(clusters) as possible not covering more than one object. By increasing the radius we consider more neighbouring pixels, directly increasing the size of the graph and lowering the amount of segments. This leads to a better segmentation but a much greater computational time.

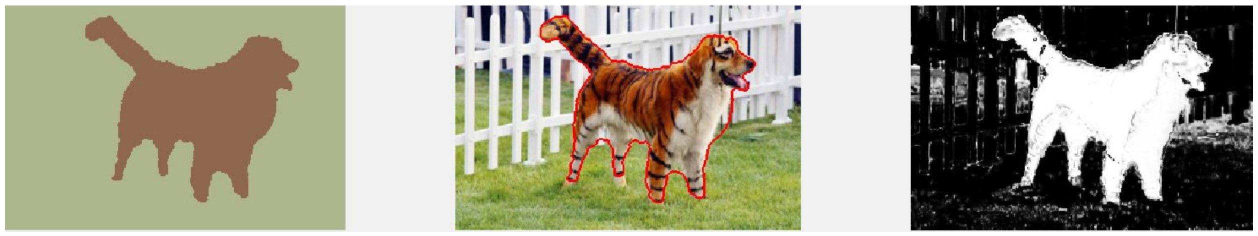
Normalized Cut problem itself is known to be NP-complete. For a general eigenvalue problem, solving for all eigenvectors runs in O(n3) time, where n is the number of pixels in the image (nodes in the graph)

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.

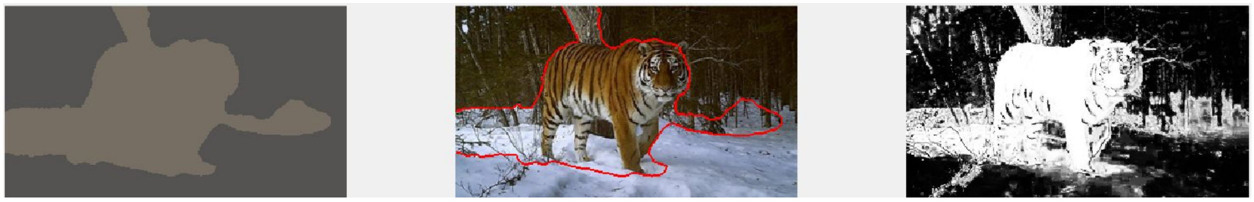
Ideal parameters for Tiger 3:

Static:
scale_factor = 0.5;
area = [80, 110, 570, 300]
K = 8;

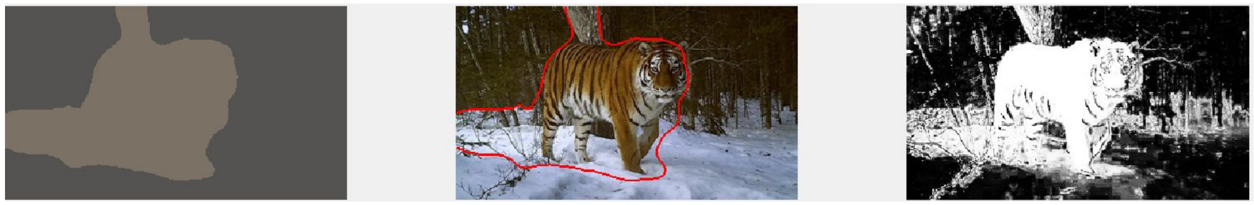
alpha = 16.0;
sigma = 18.0;



The same parameters on alpha and sigma for Tiger2 does not produce a result that is optimized for that image



Sigma controls the speed of decay for decreasing similarity of pixels, the larger sigma is, the cheaper the cost will be to cut pixels with less similarity. Alpha sets the maximum cost of cutting two entirely similar pixels. The ideal parameters for this image was
alpha = 30;
sigma = 28;

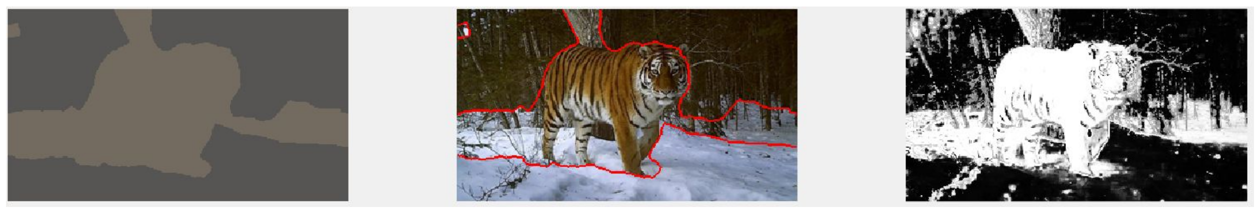


Question 12: How much can you lower K until the results get considerably worse?

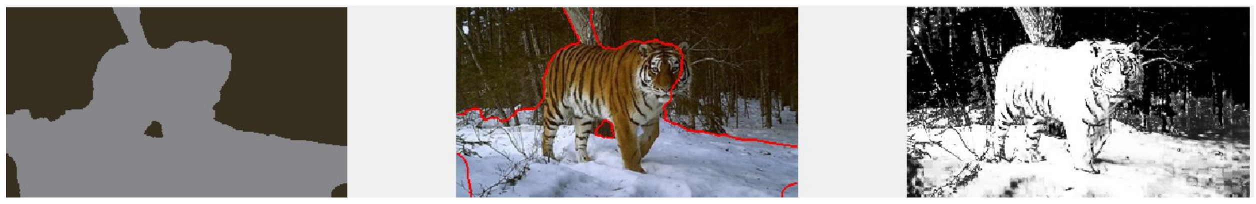
Each color in RGB is a mixture of three colors red, green and blue. The model should start to show considerably worse results if the number of gaussian mixture components are below three. Based on the theory that assumes it is possible to match any color by mixing an appropriate amount of the three primary colors RGB.

Keeping all other parameters static we vary K accordingly:

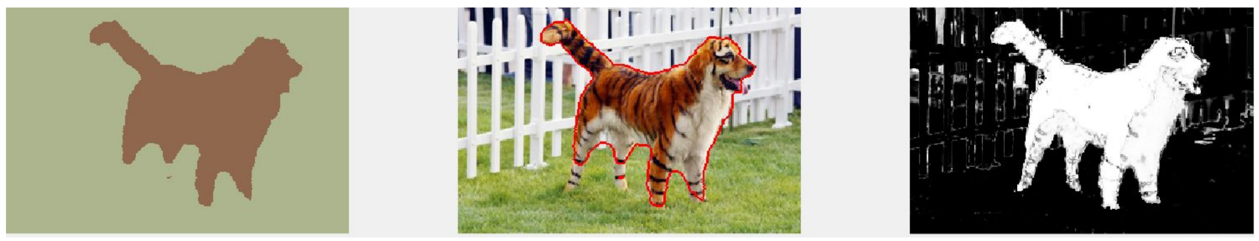
K = 3



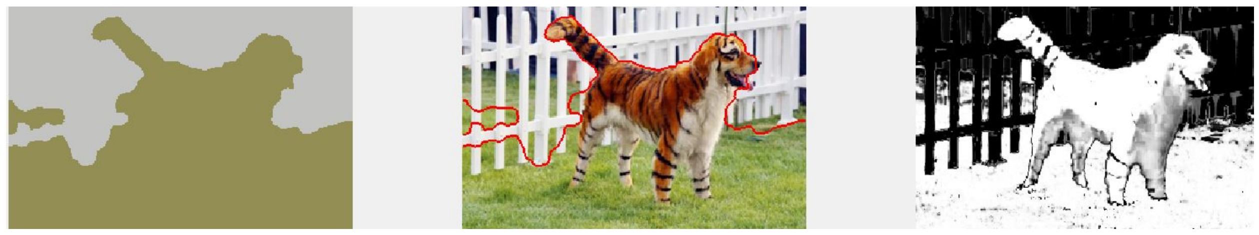
K=2



K = 3



K = 2



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!

Graph cut works well when the object of interest has an another color distribution compared to the background. If that's not the case the segmentation could be problematic. When the models for the background and foreground are very much the same the result will be worse.

If the majority of the parts of the actual foreground are concentrated within the rectangle the pixels compose a good training set. We can then be more certain that the probability of background and foreground given a certain color are more accurately calculated using the bayes rule formula.

This however is harder to do when colors of the foreground are similar to those of the background, the models for foreground and background will then be to similar and $p(f \mid c_i)$ *and* $p(b \mid c_i)$ won't be as accurately calculated for the pixel values. If the initial rectangle does not contain enough features of the object in the foreground we will have artifacts. The rectangle determines the quality of the reconstruction.

It is worth for images where we can accurately separate the foreground object from the background with a simple rectangle and the colors are different inside the rectangle from the outside.

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?Think carefully!!

- The three methods K-means, Mean shift and Normalized cut all used the idea of splitting the image into a number of segments while Graph cut utilize the idea of separating the image into foreground(Source) and background(sink).

- The number of clusters are known in K--means, The number of gaussian components in graph cut, the maximum number of clusters in Normalized cut. In the case of mean-shift we do not known how many modes will be obtained.

- K-means does not take into account position of the pixels, the location in the xy-plane, Therefore, we can not separate between two different objects that are grouped in the same cluster.
  In our case the K-means would place all pixels of the same color in the same group of superpixels whereas mean shift can distinguish objects with the same color because we work in the 5-dimensional space, taking into account the position and color of pixels.

- Graph cut and Normalized cut treat the image as a graph where each pixel corresponds to a node and set up weights on the edges between the nodes. Both of these methods try to minimize the cost of a cut by minimizing the cost and energy functions. The difference is that the Graph cut algorithm needs a prior probability that a pixel corresponds to the foreground or background to work.

- Graph cut and K-means differ in the sense that a pixel only influence one cluster in K-means whereas the pixel influence all GMMs in graph cut. Both are iterative. Clusters are spherical in K-means and ellipses in GMM.