Assignment 1 - CV

Cosmin Madalin Zaharia - i6343878

Department of Advanced Computer Sciences
Maastricht University
The Netherlands

May 2023

1 Introduction

In this paper we are going to see briefly what Mean-Shift is and the main steps of this algorithm, followed by some experiment, final consideration and how to improve it.

Mean-shift Clustering is an unsupervised learning algorithm based on clusters using sliding windows and centroids. The idea is to start around a point and apply a sphere(window) and "shifting to the mean" in an iterative way around the neighbours in order to find the centroid that presents that pixel. From this we can deduce that the number of clusters is not pre-determined but is decided during the calculation and depends on the size of this sphere.

2 Main steps

The first two steps implemented for achiving this result were:

- **Findpeak**: We starting with a point and calculate the distance between that point and all other points in the dataset. The search is limited to a spherical window with a radius r. Then, The algorithm calculates the mean of all the points within this window and shifts the window towards the area with the highest density of points, which corresponds to the highest mean. This process is repeated until the mean converges.
- **Meanshift**: The purposee of this functions is to exploit the *findpeak* function in order to assign a label to each cluster based on a simple rule. Specifically, if the distance between two peaks is smaller than half of the predefined radius r, then they are assigned the same label ortherwise a new cluster is found.

Using these two functions as they are obviously makes the code slow. To speed up the code, we use two optimizations: associating nearby data points with a peak and a converged peak, using a set() function to avoid duplicates. We collect each column of pixels around a peak in a Basin with a distance $\leq$ r/c, and finally the last points with a distance $\leq$ r around the centroid. The image is converted from RGB to LAB, modeled, and segmented along rows and columns before being reshaped to the original size and converted back to RGB.

3 Experiments

In this section we will analyse the output of the algorithm on 3 different images by changing the parameters: r, c and feature types. Obviously, the first thing we'll notice is the fact that increasing r and decreasing c makes the algorithm faster, and vice versa, decreasing r and increasing c makes the algorithm slower.

3.1 Image-1

In this first image, we have a girl leaning against a tree with a lake behind her (or a river).
Let's see how the pixels are distributed, in the interest of having an idea of how many or how the clusters are distributed:

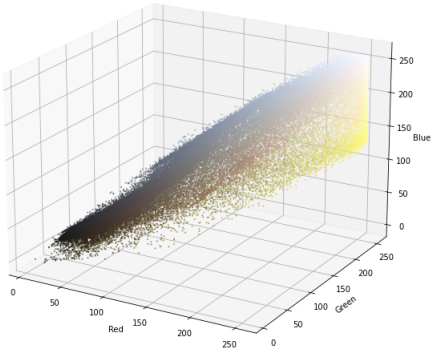


Figure 1: image-1 and its pixel distribution

Below is the table of the various attempts made with the respective values of r and c:

	c = 2	c= 4	c = 8
r =4	454.52	1943.52	//
r = 8	181.67	827.47	//
r = 12	76.12	389.34	//
r = 16	82.34	366.17	698.41
r = 30	27.45	422.77	591.14

To follow, I have selected some of the processed image (you can find the other in the plots_directory) that show the evolution of the segmentation process.

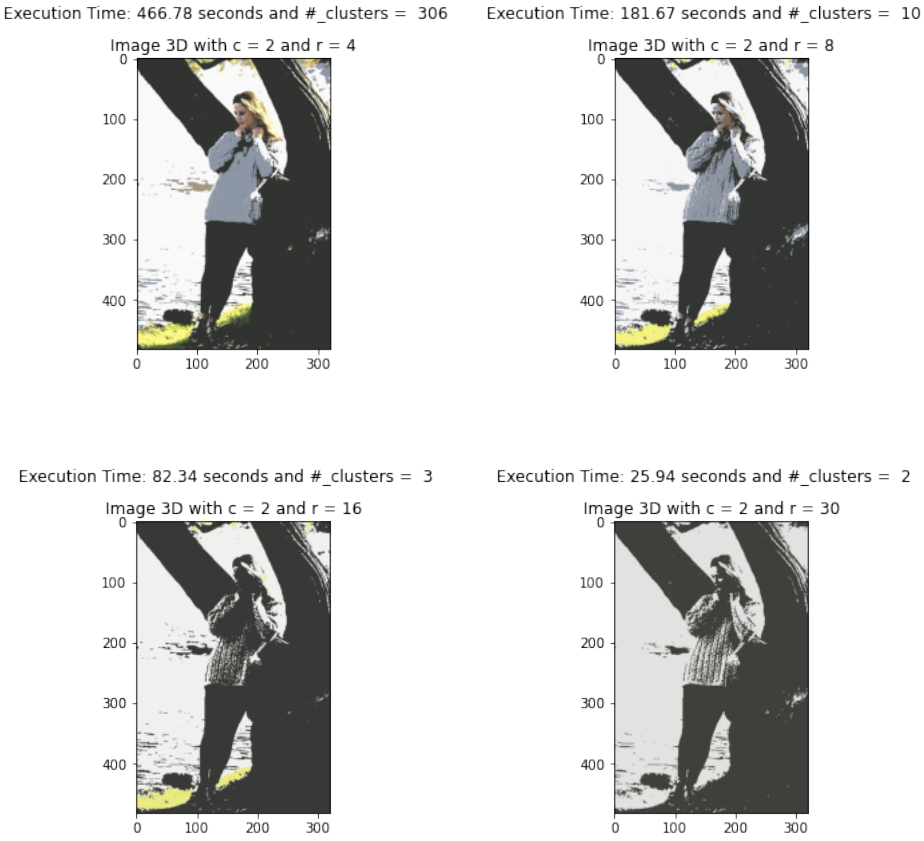


Figure 2: 3D image segmentation of the first image based on r

We can immediately see that with r = 4 we are very close to the original image with 306 clusters, but with r = 8 have we drastically reduced the number of clusters.
We can also see that the less "dominant" colours tend to be absorbed by the dominant one. Finally, only with r = 30 have we lost one of the three clusters.
In the image folder you will find other processed images with different c, they are almost similar, just with a slight change in brightness.

Now let's have a look to the the processing in the 5D feature space, where in addition to the channel color RGB we also include the position of each pixel:

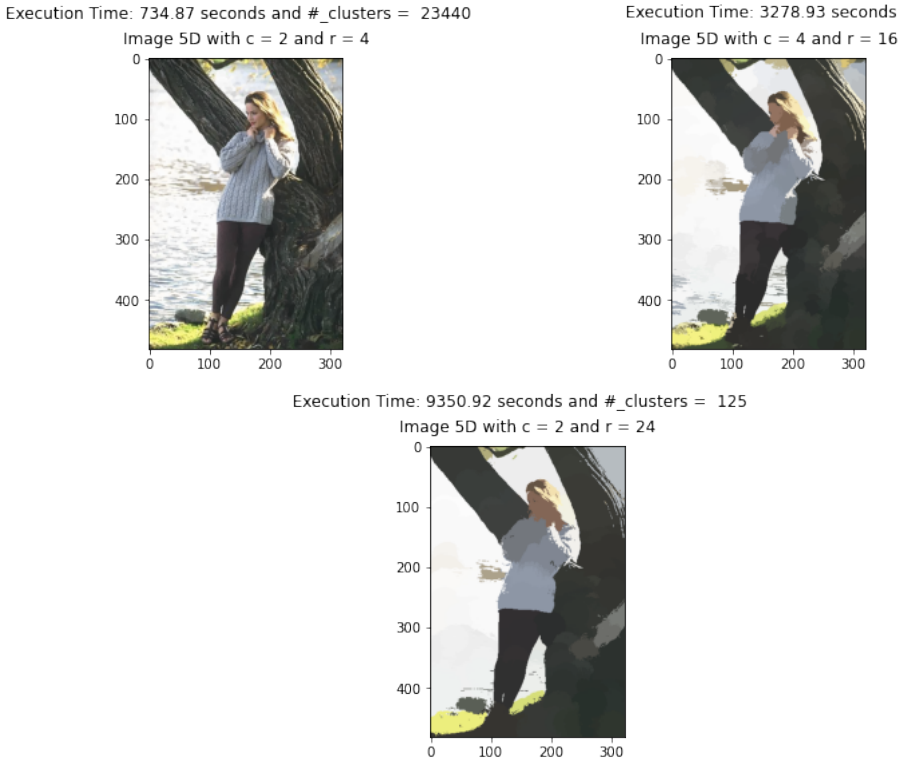


Figure 3: 5D image segmentation of the first image based on r

What you can see in this case is that, unlike before, here the segmentation is more clear (in fact it almost looks like a painting) that because in the previous case a pixel of a colour or similar belonged to the same cluster, but

now not only the colour is important but also the position, so even if two pixels are closer but in a different position, the merge is different. In fact, what I like here is that even though we lose a lot of detail, we have a clear separation of each object, especially for the face of the girl, which in the 3D case has become part of the tree with a big r. Lastly, you can also see that in the 3D case, increasing r doesn't necessarily mean losing the image, but preserving the important elements of that image.

3.2 Image-2

In this case we have a masked person and a background with different colours, tending towards darkness and I think that the wall and the grass tend to be part of the same cluster since as they seem quite "similar".

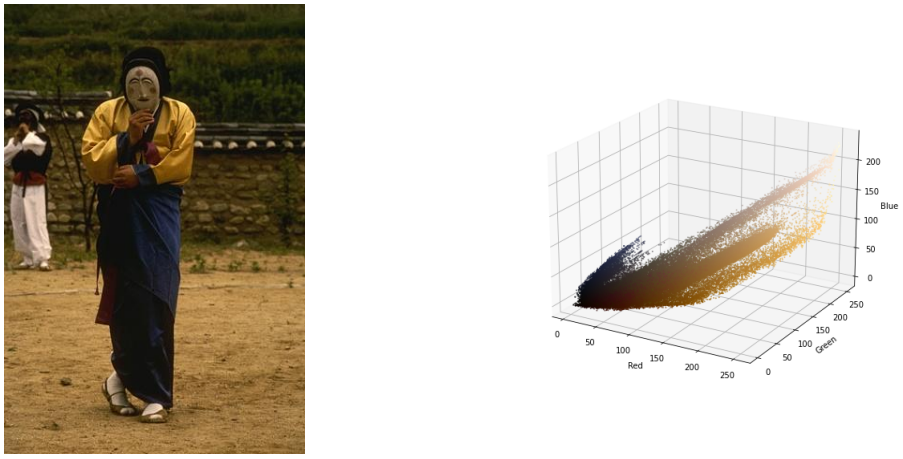
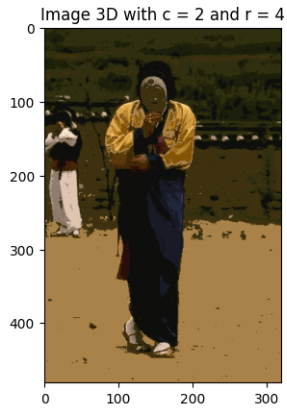


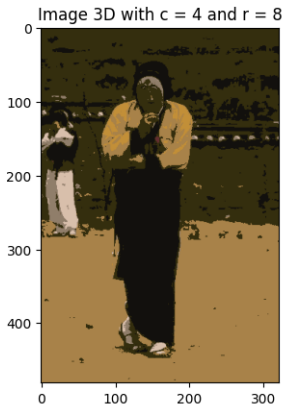
Figure 4: image-2 and its pixel distribution

With respect to before, we have a greater concentration of pixels around dark and all pixels have some kind of "connection" between them. Let's see the 3D result:

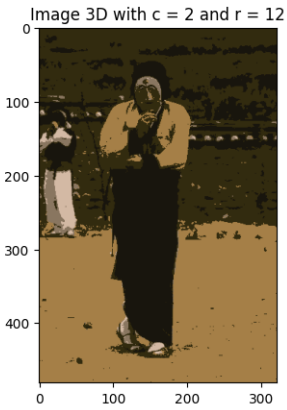
Execution Time: 2488.65 seconds and #_clusters = 92



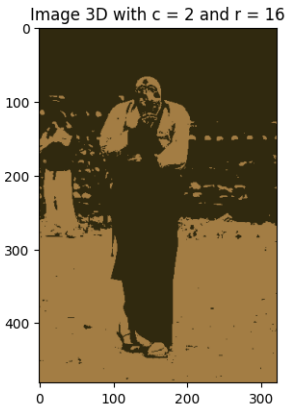
Execution Time: 5905.53 seconds and #_clusters = 8



Execution Time: 340.98 seconds and #_clusters = 5



Execution Time: 362.34 seconds and #_clusters = 2



These images confirm what I had previously guessed, namely that dark colours tend to merge into a single cluster, ditto for light colours. Additionally, it's interesting that the white is present since the beginning until is absorbed by the "brown" cluster (the same for the gray) instead by the the black.

Now let's have a look to the 5D case:

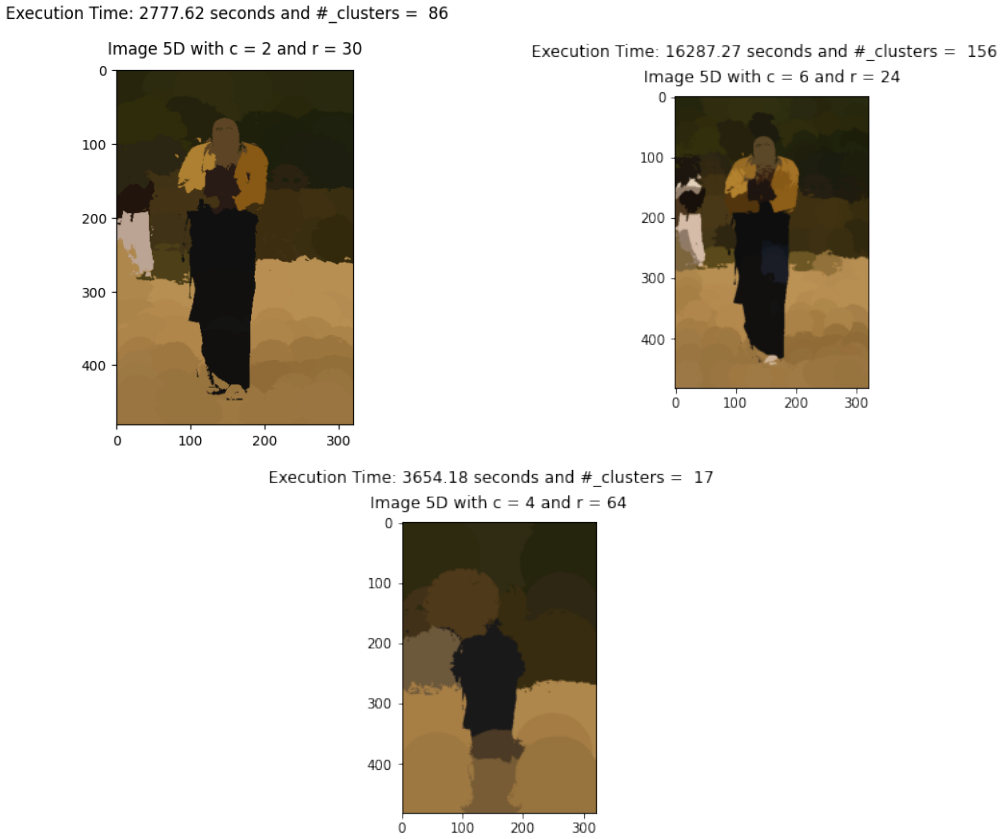


Figure 5: 5D image segmentation of the second image

As you can see, the number of clusters remains high even when the window size is large. Also with regard to the 3D case, the person behind loses a leg and it becomes quite difficult to understand the separation between the wall and the grass. Indeed, this is one of those cases where the 5D case is not good to use respect to the 3D case.

3.3 Image-3

In this last case, we have the Big Ben(?), trees and two policemen. The interesting fact that I noticed when I ran the algorithm on this image is the speed with which it works compared to the first two, for instance fast with a small r while slow as r increases (even with a small c).

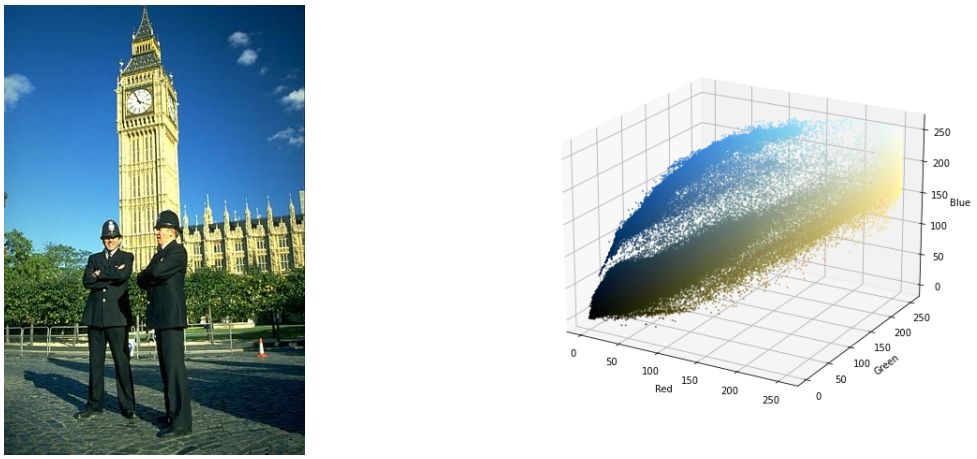
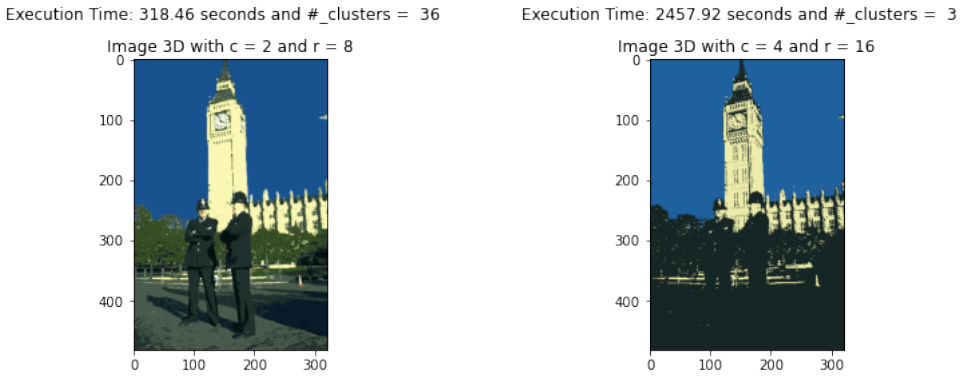
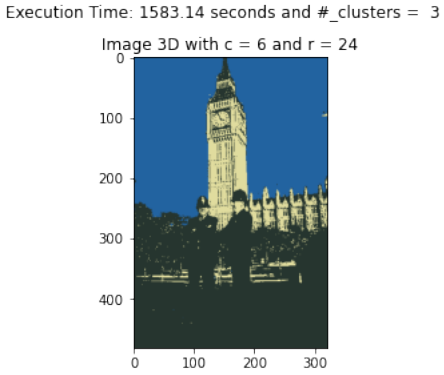
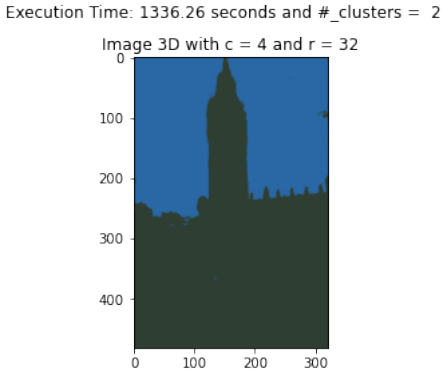


Figure 6: image-3 and its pixel distribution

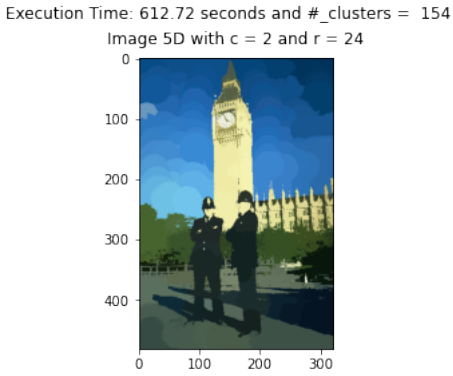
I'd like to draw your attention to the distribution of the pixels, in fact you can immediately see that there is a strong division between two types of colour, blue and dark-brown, which means that we need a huge r to keep these two different shades of colour in the same cluster.

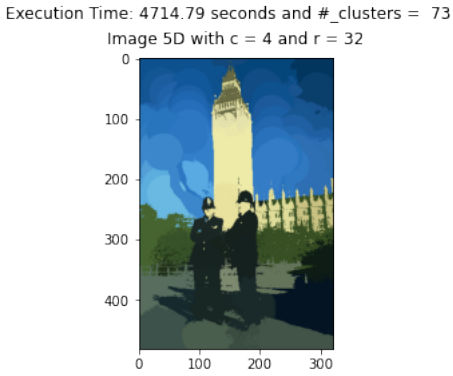






As I mentioned before, given this pixel distribution the image keeps to be clear also with an high r. Lastly, even if r is high the delineation of some object is perfect.





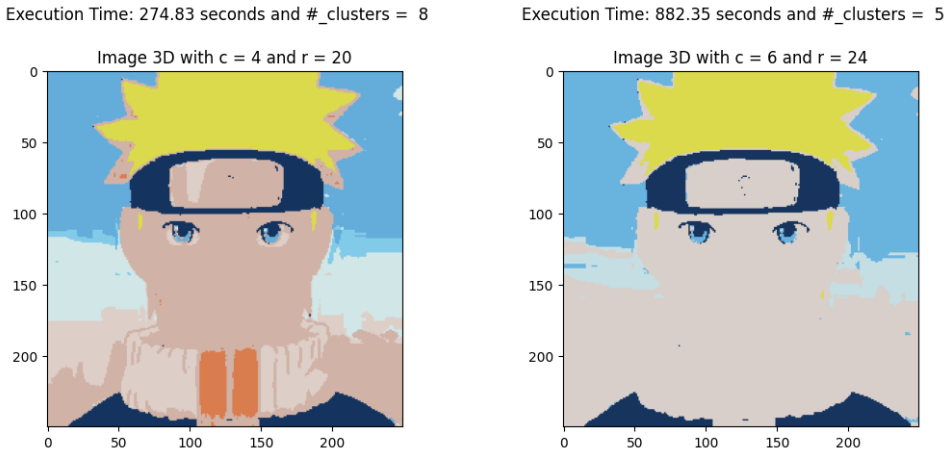
In this case the loss of detail is very low but easily recognisable as with a simple r=4 we have lost some of the details of the big ben but overall the image is very similar to the starting image. Beside that, if you look at the same image with r=32 and c=4 the difference it's huge only for little details like the texture fo the Big Ben.

4 Final observation

We started this paper by saying that in general increasing r and decreasing c makes the code run faster, but in the last example we saw that this is not true for every image. Also we thought that the 5D segmentation was better than 3D segmentation but this is not true, take the second case as an example.
However, what made the difference, and safe time on doing the predictions, was seeing and analysing the distribution of pixels in an image, in fact, by looking at these distributions you can get a general idea of how many clusters there might be, or whether it is more appropriate to use a 3D than a 5D segmentation.
In addition to analyzing pixel distributions, it may also be necessary to consider other actors such as noise levels, image resolution, and the presence of overlapping objects in the image.
To sum up, when selecting the radius value for image segmentation, it is important to keep in mind the specific details of the image and the level of segmentation required. For instance, if the goal is to segment only the basic shape of a person or a building, a larger radius can be used as it requires less time and produces fewer peaks that distinguish the image elements from their background. However, for more detailed segmentation, such as in the case of clothes, Big Ben or distinguishing between a masked person and their background with different colors, a smaller radius would be more appropriate. Ultimately, the choice of radius should be based on the analysis of the image characteristics, such as pixel distributions, colors, and texture, to achieve accurate and efficient segmentation results.

5 Other experiments

In this section I want to propose two different cases in order to have a complete analysis of how the algorithm works on other type of Images, in one case I used a character from an anime to see how it deals with digital image and a picture did by my phone and resized through python (you can find the code in the notebook called image_resize). In fact, as you may noticed in some cases like in the second image, the dark color tend to be part of the same cluster, in this section I want to see if the same is valid for light color.
Let's start with our character "Naruto Uzumaki" (you can find the original one in the image_plots folder):



You can see how strange this is because, as I said, there is a fusion between the light colours as in the second image (for the dark ones), but I thought that since the image has a kind of division as in the third image, the face would be "safe" and this doesn't seem to be the case and the reason may be the fact that in the case of the third images the police and the trees had a completely different gradation of colour. It's also interesting to see how, with just a little different r, the pink, which was the dominant colour in $r \leq 24$, is absorbed into the grey cluster.

The following is an image I took in Maastricht in the West-North part. The focus here is firstly to understand better how the algorithm deals with the shadow, and secondly to see if it would be a merge from the beginning as in the second image, or a division as in the first, which tends to disappear as r increases.

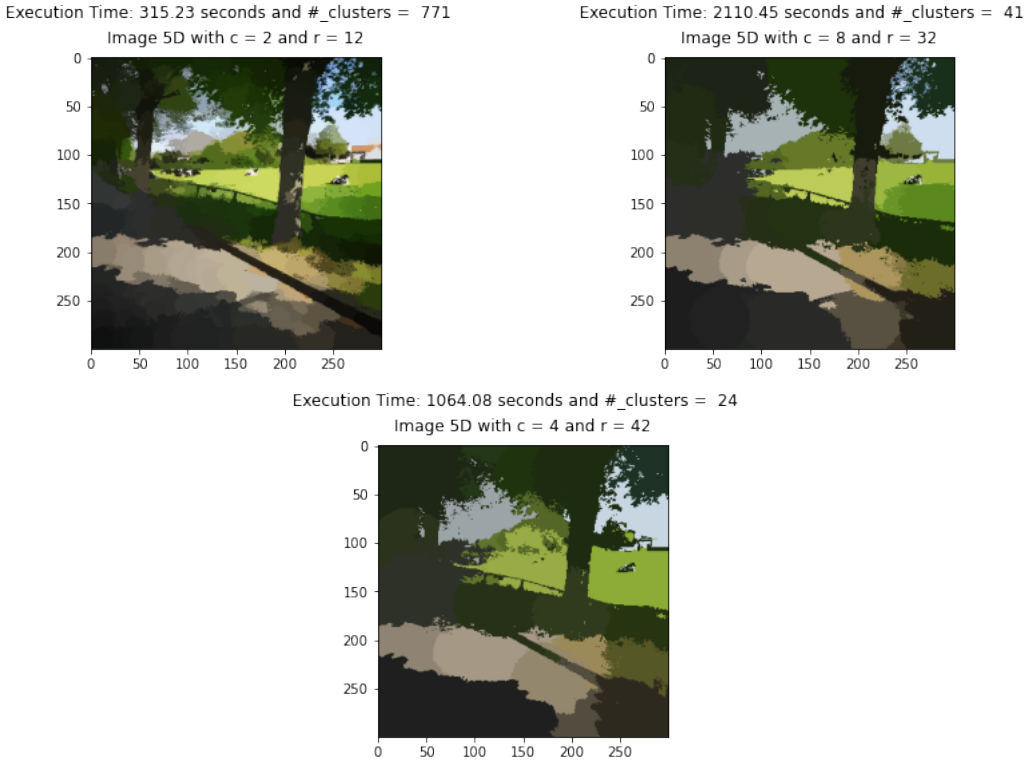
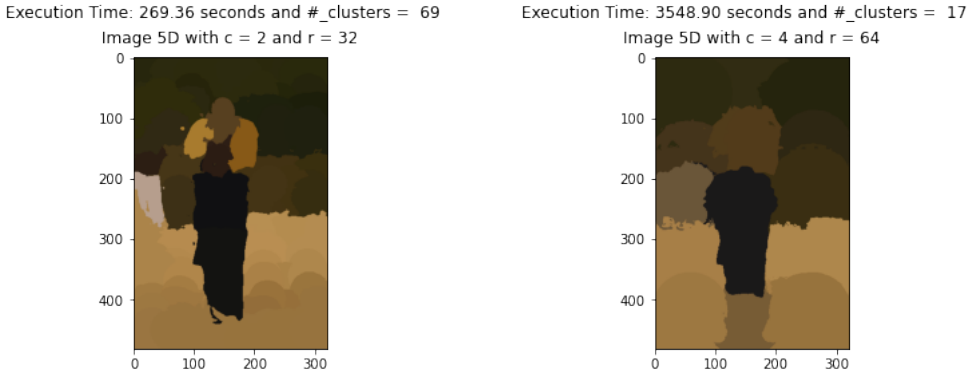


Figure 7: 5D segmentation

Unlike the other experiments, increasing r doesn't effectively reduce the computation time and regarding to the shadows the algorithm cope them quite well. Surprisingly, even with a large r the cow in the right foreground did not disappear while the house behind did.

6 Possible improvements

In this section I want to talk about two improvements/approaches that someone can try to improve the algorithm.
The first, regards the original image that sometimes might be too precise or with noise, so the idea is to apply before the segmentation a filter. In this case, for space reason, you'll see the effect of applying the algorithm on a blurred image.
In this case I've applied a Gaussian filter with kernel size 5 and standard deviation 2.



The result is quite good, as we can see the execution time is significantly reduced in a case and little in the other. In a working context, the idea should be that based on the image in front of you, you decide on the filter that best suits the situation, so that you can significantly reduce each time computation.
The second idea was that when searching for the centre of mass, instead of using a sphere where all points have the same importance, you can try to use a Gaussian search window. Unfortunately this idea didn't work very well, I tried some changes and combinations for the sigma, but the time was too high, the tqdm library suggested more than 38h on avarege.