Maastricht University

Computer Vision

Assignment 1: Mean-shift

Maastricht University

Faculty of Science and Engineering

Department of Data Science and Knowledge Engineering

Authors

Martyna Sylwia Mikos, i6155316

1 Introduction

Mean-shift is a non-parametric algorithm that performs feature space analysis and clustering by iteratively associating datapoints to the highest probability density of the dataset (peak) in the giver region. Since non-parametric methods don't have embedded assumptions about the feature spaces to be tested, the advantage of this algorithm is that the number of clusters is determined by the algorithm and doesn't need to be predefined.

In this assignment mean-shift algorithm was implemented and optimized in two steps. Two functions were implemented for this algorithm: $findpeak$, that finds the density peak associated with each datapoint and the main function: $meanshift$ that calls $findpeak$ and assigns the label to each datapoint while storing the peaks.

Above functions incorporate two speedups. One of them is based on basin of attraction that associates the data point with its cluster withing a window of radius r. The second depends on the fact that datapoints on a search path within a a radius of r/c are associated with the converged peak. Both functions were tested on provided dataset containing data from two 3D Gaussians and three different images.

Lastly, a function $imSegment$ was implemented to perform image segmentation in order to group the pixels with the same attributes and extract the shapes of the objects. In order to do that the RGB image needed to be converted to L*a*b color space, which describes all the colors interpretable by a human eye. This is because the difference between colors is measured by Euclidean distance. In other words, the distance perceived by a person correlates with the distance in space. The next step it to call the function $meanshift$ on the converted image to extract the labels and peaks associated with pixels. After that the pixels are recolored according to peak values using labeled image. Both, 3D (color) and 5D (color + image location) features were implemented and tested.

2 Mean-shift implementation

Implementation of the $meanshift$ was followed by implementation of a function $findpeak$ that searches for a center of the cluster (peak) by calculating a distance between one data point and all the other points in the dataset. The search region is a predefined spherical window of radius r. Algorithm computes the mean of points lying within that window and shifts this window towards the area with the highest density of points (the highest mean) eventually converging under a threshold (assumed t = 0.01) to the centroid of the cluster - a density peak.

Function $meanshift$ calls the $findpeak$ on the datapoint and assigns the label of the cluster. Each time a new peak is found (peak is considered new if the value of the peak found by $findpeak$ is larger then the value previously found by $r/2$), the new label is created and assigned to the points within that cluster.

First implementation is not very time efficient therefore two speedups were introduced:

Mean-shift first speedup ($meanshift\_opt$)

Instead of going through every single point of the dataset, we can instantly label those points that lay within one window size of of the peak. This works, because points laying close to the peak are most likely going to converge to it, so there is no need to loop through all of them because they can be associated with the closest cluster centroid up front. In order to do that $meanshift\_opt$ calls $findpeak$ function that outputs $found\_distance$ vector that contains indices of the datapoints at the radius r from the associated peak.

Mean-shift second speedup ($meanshift\_opt2$)

The second speedup takes into account not only the points close to the peak but also the points within a distance of r/c (with c being a constant) of the search path. These points that are found on the path are associated with the converged peak.

$meanshift\_opt2$ calls $findpeak\_opt$ function that uses a checkpoint vector ($cpts$) storing a 1 for each point that is a distance of r/c from the path, 0 otherwise. Then finds the indexes for which the value of $cpts$ is 1 and automatically labels the points found on the search path with the same label.

3 Segmentation implementation ($imSegment$)

This function performs image segmentation by running $meanshift\_opt2$ on the transposed, flattened image. The input image needs to be reshaped and converted to Lab color space in order to have a desired shape for the meanshift input. Using labels, a matrix is created with each pixel assigned to a cluster. Peak values are used to color the pixels in an image based on the labels. Two different types of features were used:

1. Lab color values (3D feature vector)
2. Lab + (x,y) coordinate values (5D feature vector)

Afterwards the segmented image is reshaped to the original size and converted back to RGB color space.

4 Experiments

4.1 Function evaluation on the dataset consisting of two 3D Gaussians

Before implementing segmentation function $meanshift$ was debugged on the provided dataset consisting of two 3D Gaussians, expecting two different clusters. Figure 1 shows a plot of the clusters based on the found peaks and labels. Based on that figure clustering was indeed successful.

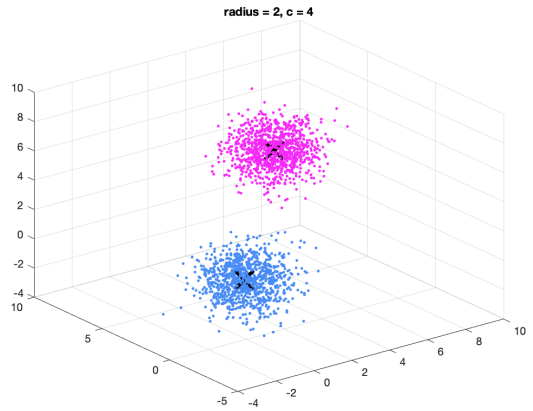


Figure 1: Cluster plot

Table 1 compares the time efficiency of the $meanshift$ functions. First speedup already showed a significant improvement being 1 second faster than the non-optimized function. Second speedup is roughly 10 milliseconds faster, however it makes a huge difference with a bigger datasets like RGB images.

Function	Elapsed time (seconds)
$meanshift$	1.686675
$meanshift\_opt$	0.527674
$meanshift\_opt2$	0.413883

Table 1: Execution time of $meanshift$ function and its speedups

4.2 Function evaluation on images

Function $imSegment$ was run on three different images in order perform segmentation. Both features were used for each image - 3D and 5D, with varying radius r and constant c to explore their impact. In order to plot the clusters function $plot3dclusters$ was modified to $plot3dclustersRG$ which instead of using random colors, uses the real RGB values extracted from the peaks.

4.2.1 Image 1 ($img1$)

First experiment was run for two different values of c and six different radius values. Time was measured for each parameter combination and presented in Table 2. It is pretty straightforward that the bigger the radius the faster the segmentation and naturally the bigger the c the slower the algorithm since it reduces the radius size of distance in the search path.

	c = 2	c=4
r = 4	228.95	694.62
r = 8	42.96	187.79
r = 12	14.49	71.33
r = 16	7.24	35.597
r = 20	3.91	20.90
r = 24	2.92	13.21

Table 2: Execution time for $img1$ with 3D feature type, changing radius

Results of the segmentation for the 3D feature space are presented in Figure 2 and results for the 5D setting are shown in Figure 4.

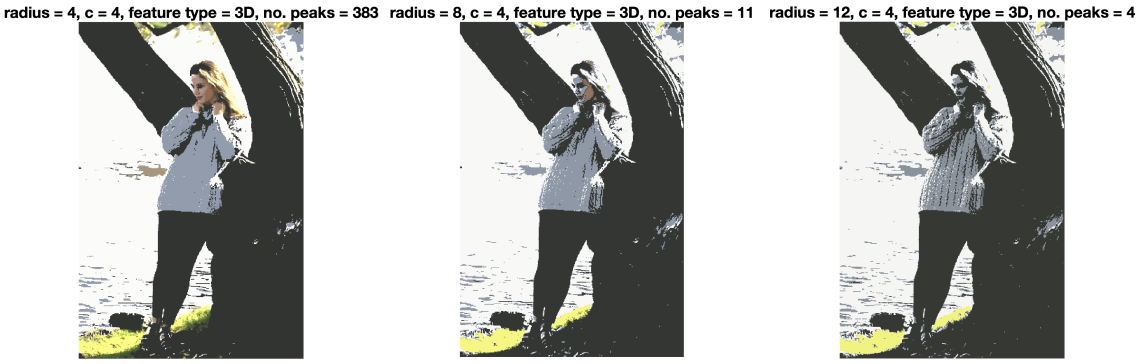


Figure 2: Segmentation of $img1$ with 3D feature type with varying radius for $c = 4$

Naturally, the smaller the radius the bigger the number of clusters found and more detailed the image. In Figure 2, $[r = 4; c = 4]$ setting resulted in an image that is very close to the original one and acquired 383 clusters. On the other hand, the image with $[r = 12; c = 4]$ setting represents only 4 peaks. In order to visualize the clusters, a point cloud was plotted and displayed in Figure 3 that shows how the size and the number of the clusters changes wen we decrease the radius. For smaller radius we have a larger color variety that decreases with radius because the similar and less dominant (close) colors are merged into one of the clusters.

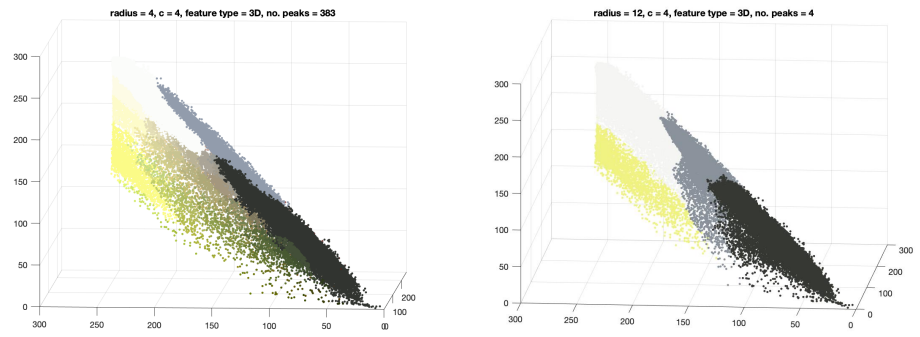


Figure 3: Cluster cloud-point of $img1$ with 3D feature type with varying radius for $c = 4$

The next experiment was performed in 5D feature space, which means we do not only have the

information about the color in R, G and B layers but also the position of each pixel. As it can be seen in Figure 4 it changed the segmentation a lot. Previously a pixel with a certain color value was associated with the ones having similar values, regardless of its position in an image. In other words two distant pixels with the same color belonged to one cluster. Now, the peaks are associated with the color and the position of the other pixels in an image. Therefore, two pixels with a similar color values but different place in the image won't be associated with the same cluster. That is why the result of the segmentation is clear color and shape separation and larger amount of peek. This becomes obvious by looking at the segmented images in this and the next experiments.

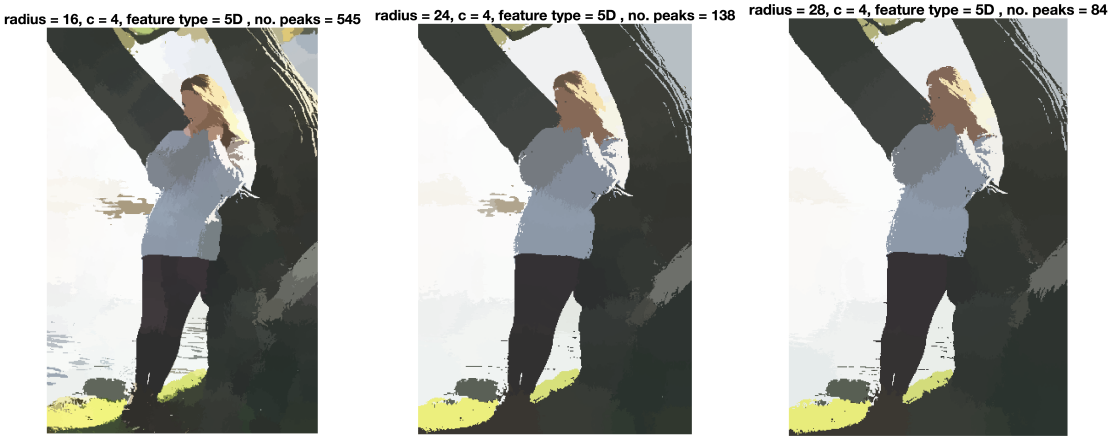


Figure 4: Segmentation of $img1$ with 3D feature type and varying radius for $c = 4$

4.2.2 Image 2 ($img2$)

In this section the main focus was put on analysing different values of c in different feature spaces. Algorithm was run for $c \in \{1, 2, 4, 6\}$ and some of them were shown below. The images that didn't demonstrate any additional findings were not presented in the report.

In Figure 5 the difference between $c = 4$ and $c = 6$ is barely noticeable. Difference between $c = 1$ and the remaining images is mostly in the background, where the segments on the ground are smoother and have more regular shapes with bigger c. Moreover, one might have an impression that segments are a bit more clearly differentiated.

In Figure 6 represents the same image with different radius. Here, the difference is a lot bigger. Segmentation with $c = 1$ resulted in segments that are not as accurate and some of the datapoints were inaccurately classified, which with such big radius could be expected, although increasing c improves the segments significantly.

Lastly, the experiment run in 3D settings confirms the above statement that higher c improves the segmentation. In Figure 7 the difference might be small but segmentation is in deed improved.

Figure 5: Segmentation of $img2$ with 5D feature type with varying c for radius $r = 20$

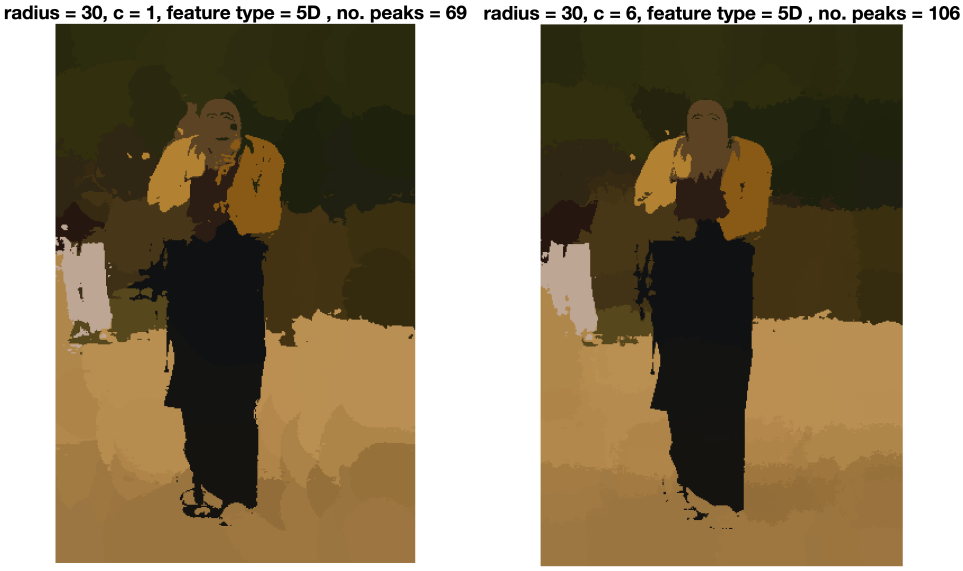


Figure 6: Segmentation of $img2$ with 5D feature type with varying c for radius $r = 30$

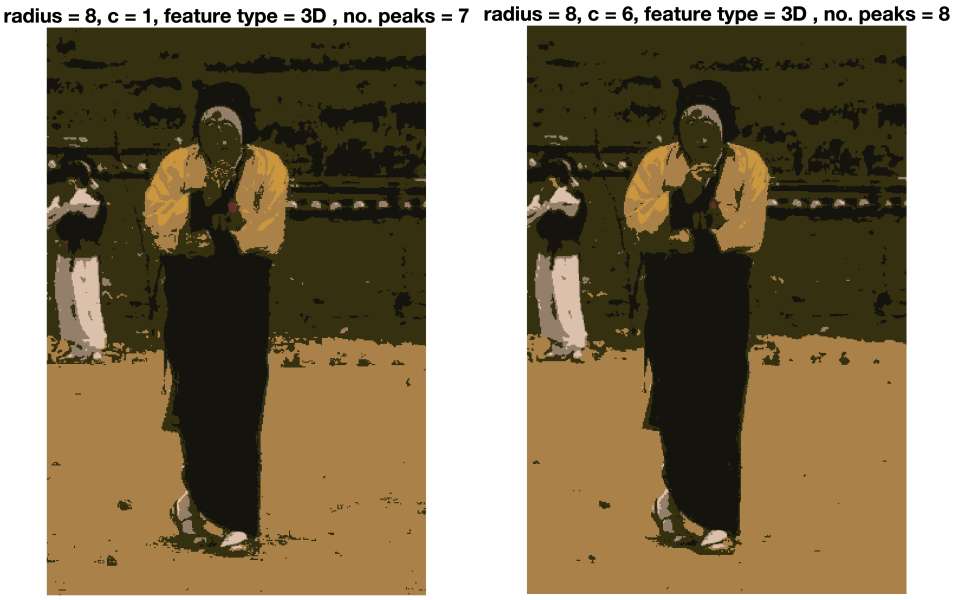


Figure 7: Segmentation of $img2$ with 3D feature type with varying c for radius $r = 8$

4.2.3 Image 3 ($img3$)

Last experiment was run to verify the observations made so far on another image. Here, for each feature type two values of radius and two values of c were tested.

In this case, the difference if very hard to spot, although in Figure 10 it can be clearly seen that higher c resulted in a bit more detailed segmentation. Similarly, in the Figure 11 that shows the 5D segmentation with the higher radius, the Big Ben also has more details and more different colored clusters can be spotted in the background on the sky.

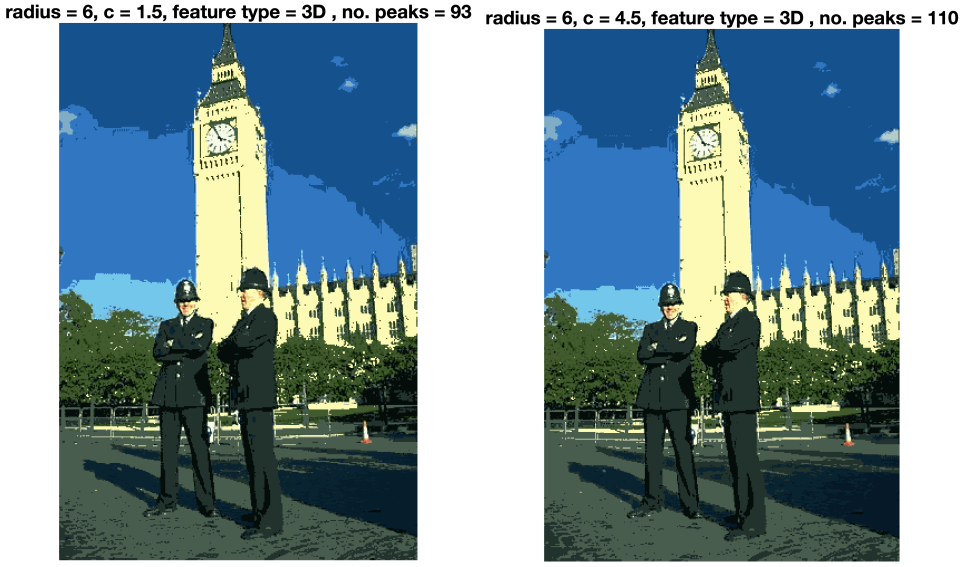


Figure 8: Segmentation of $img3$ with 3D feature type for $r = 6$

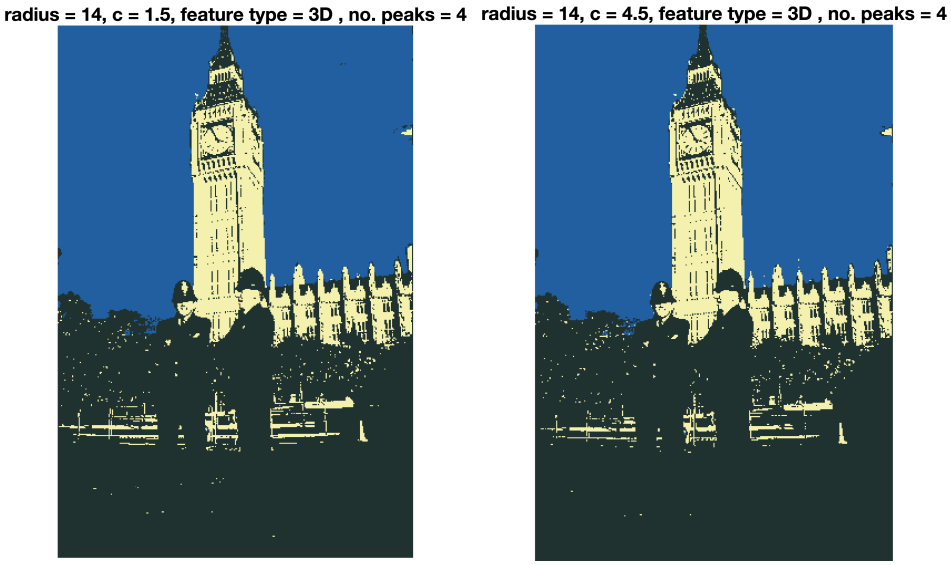


Figure 9: Segmentation of $img3$ with 3D feature type for $r = 14$

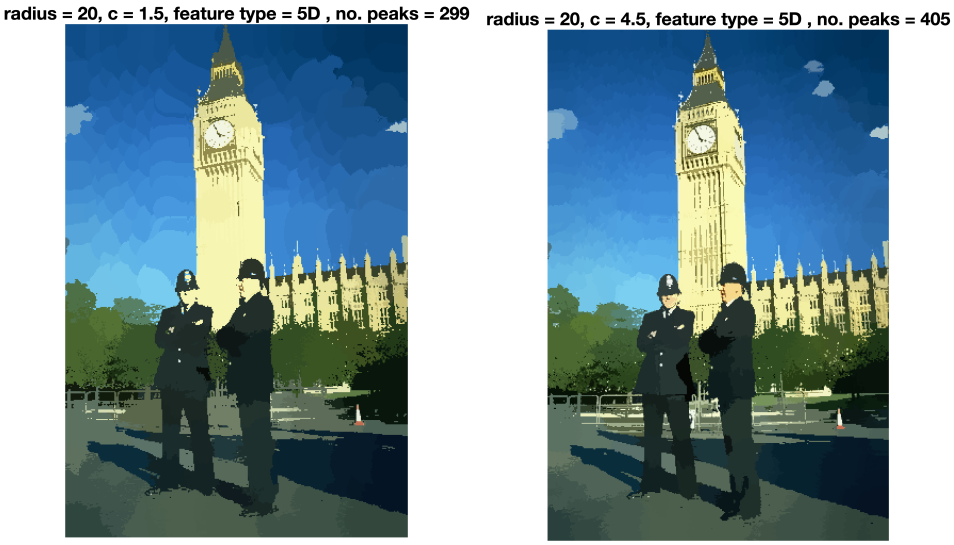


Figure 10: Segmentation of $img3$ with 5D feature type for $r = 20$

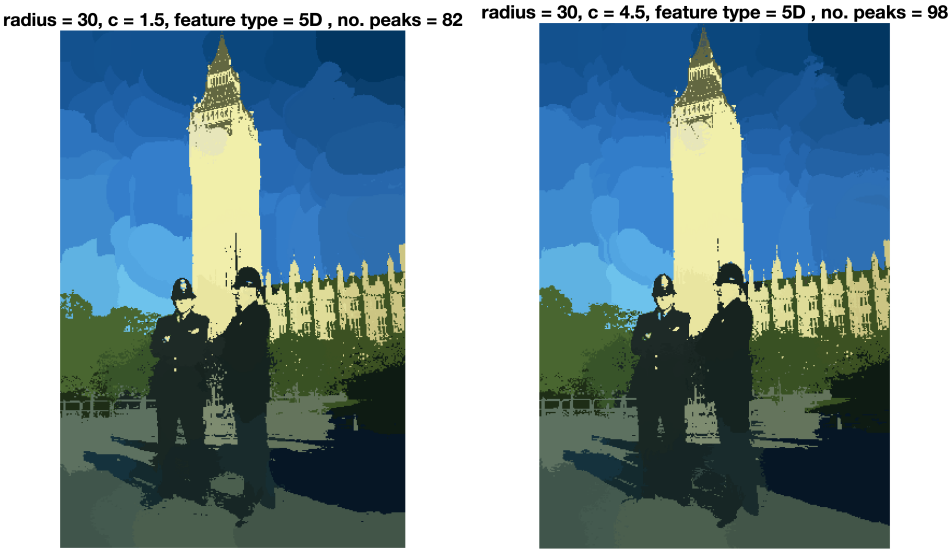


Figure 11: Segmentation of $img3$ with 5D feature type for $r = 30$

5 Conclusions

Overall, increasing c provides more number of peaks therefore a more detailed segmentation. However it also takes significantly larger amount of time. For $img3$ and $r = 20$, for $c = 1.5$ the algorithm execution time was 2.7 minutes. Whereas for $c = 4.5$ the segmentation took 31.6 minutes. It without a doubt significant increase in time and the effects are a lot of times barely noticeable, especially for a small radius in the first place. From the experiments it can be concluded that c being around 2 is usually the most optimal parameter in terms of time and results ($img3$ with $[r = 20; c = 2.5, 5D]$ took roughly 10 minutes to execute.

The optimal radius is of course image-dependent. Although using a very small radius, like $r = 8$ in 5D segmentation (or $r = 4$, $c = 4$ in 3D in Figure 2) detected 4398 peaks, which is not really a goal of segmentation since its basically restored the original image with clusters. Moreover the choice of r should be dependent on the amount of clusters we want to get, i.e. how many features we want to detect. If our goal is segmenting just a shape of the person or a building than we would use a big radius since it takes less time and provides with only few peaks that distinguish image elements well from its background. For more detailed segmentation, like clothes of a Big Ben clock we should naturally use smaller r.

Figure 12: 5D segmentation with small radius

6 Preprocessing steps

There are a lot of different possible preprocessing steps that can be performed on the image in order to improve segmentation. Few of them were tested and compared with the segmented images without preprocessing to evaluate the performance.

Preprocessing tools used are:

- Image Normalization, which changes the range of pixel intensity values and can improve images with poor contrast. It can increase contrast for improved feature extraction.
- Median Filter is used for removing the noise from the image (like salt and pepper) while preserving edges if the image.
- Gaussian filter (with smoothing kernel with standard deviation of 1) is used to remove noise, reduce details and enhance the structure of the image.
- Wiener Filter (with mean and standard deviation of 5) is an adaptive noise-removal filtering, used for restoring an image that contains blur and noise.

The results of segmentation on the preprocessed image vary greatly depending on the characteristics of the image itself. Therefore, order to obtain desired results we need to adjust the method accordingly.

For instance, noise considerably influences the quality of segmentation because it disrupts the useful information that are needed to be extracted from the image like segment boundaries. Therefore for the noisy images the nose reduction filters and methods can help improve the quality.

Below couple of experiments were run to exploit methods mentioned above on the 3D and 5D settings.

Figure 13 and 14 shows how the prepossessing effects on the original images. We can see that the colors after applying Gaussian and Median filtering changed which would probably affect the values of the peaks themselves because they contain the color values.

Figure 13: Preprocessed $img1$

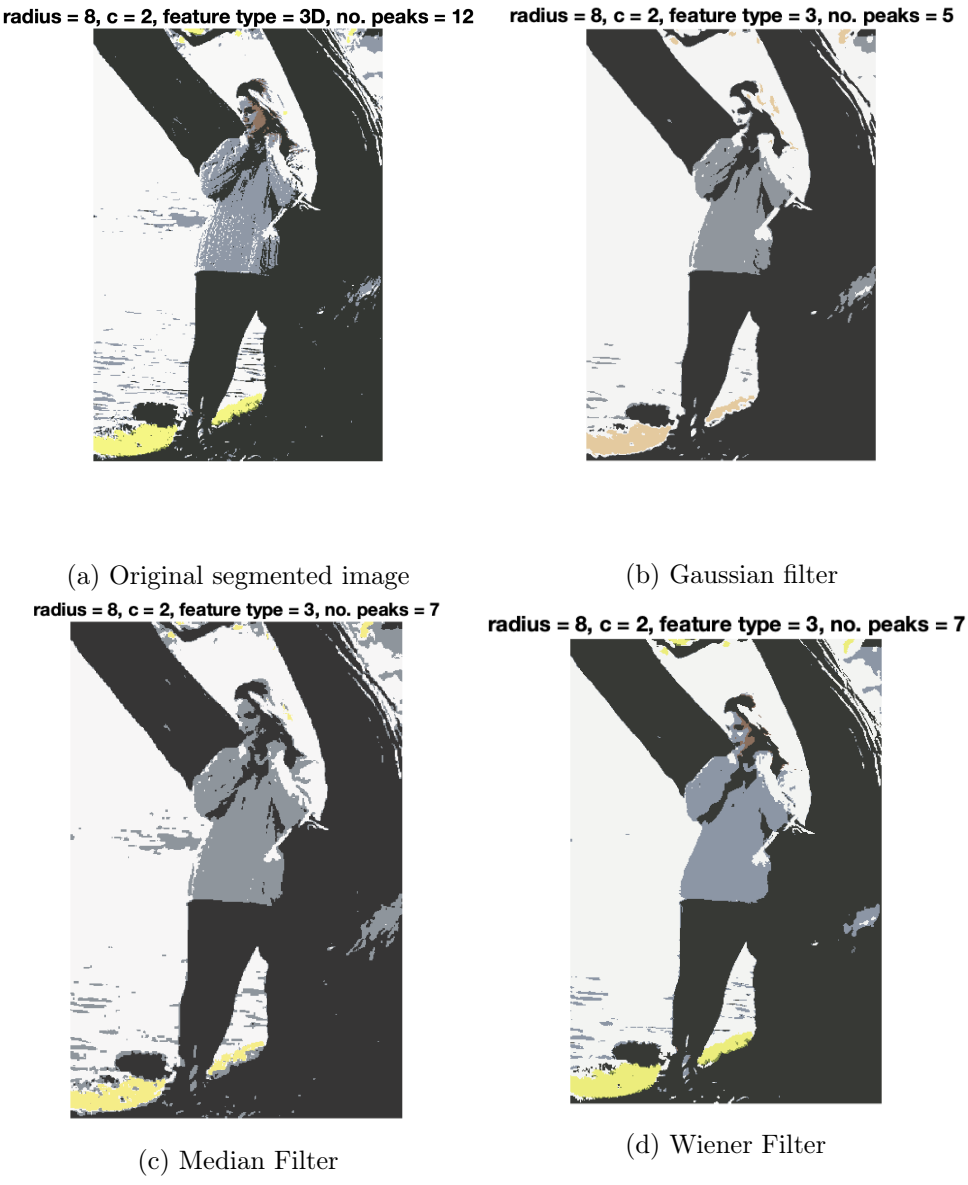
Figure 14: Preprocessed $img3$

According to the Table 3 in all the cases the time for segmentation decreased a lot, especially in case of the Gaussian filter that improved segmentation tremendously in terms of time efficiency. This is not a surprise since it is not only removing noise (like media filter) but also blurring the edges, which speeds up segmentation process.

Figures below show the segmented preprocessed images. All images were normalised before applying filters. In case of $img1$ Wiener filter performed very well resulting in well shaped segments. For $img3$ the Gaussian filter worked pretty well but the Wiener filter didn't make a big difference, which could be caused by the choice of parameters.

*img*1	Time (seconds)
Original segmented image	43.7
Gaussian filter	6.8
Median Filter	26.2
Wiener Filter	37.6
*img*3	
Original segmented image	162.7
Gaussian filter	101.5
Median Filter	154.1
Wiener Filter	144.6

Table 3: Time elapsed for segmentation



(a) Original segmented image

(b) Gaussian filter

(c) Median Filter

(d) Wiener Filter

Figure 15: *img*1 preprocessing in 3D feature space for r = 8 and c = 2

(a) Original segmented image

(b) Gaussian filter

(c) Median Filter

(d) Wiener Filter

Figure 16: *img*3 preprocessing in 5D feature space for r = 20 and c = 1.5