Assignment 1 - the Mean Shift clustering algorithm

Computer Vision

Kirill Shcherbakov (i6248484)
Maastricht University

May 17, 2021

1 Introduction and Overview

1.1 Algorithm description

One of the powerful clustering algorithm used in Unsupervised Learning is the Mean Shift algorithm [1]. It does not make any assumptions and hence it is a non-parametric clustering technique which does not require prior knowledge of the number of clusters, and does not constrain the shape of the clusters. The Mean Shift algorithm assigns the data points of an n-dimensional data set by iteratively associating each point to the highest probability density (peak) of data points. Despite a number of advantages listed above, this algorithm also has its drawbacks:

- The Mean Shift algorithm does not work well in case of high dimension, where number of clusters changes abruptly.
- We do not have any direct control on the number of clusters. In some applications (e.g. figure-ground or medical image segmentation) the user may seek a specific number of clusters.
- It cannot differentiate between meaningful and meaningless modes.

1.2 Work overview

In this work, the Mean Shift algorithm was implemented and optimized, on the basis of which an application for image segmentation was developed. The implementation of the Mean Shift algorithm consists of two functions:

- findpeak - performs the peak searching processes associated with each point.
- meanshift - calls findpeak for each point and assigns a label to each point according to its peak.

The naive implementation of the Mean Shift algorithm is slow. Therefore, two speed-ups are incorporated:

1. The first one is known as the "basin of attraction", and it associates each data point that is at a distance less or equal r.
2. The second one is based on a fact that points that are within a distance of r/c are associated with the converged peak.

All implementations are tested and debugged using the provided data set, which stores a set of 3D points belonging to two 3D Gaussians, and the results are plotted using the given plot3dclusters function.

For the actual segmentation, the function imageSegmentation is implemented. In this function, the image data is converted from the RGB space color to the LAB space color in order to allow the Euclidean distances to correlate better with the human eye color changes. Apart from that, a new feature that allows to include spatial position information is defined in this function.

2 Implementation details

The entire implementation has been done in the Python programming language. Following the recommendations of the task provided, the following functions have been implemented: findPeak, findPeakOpt, meanShift, meanShiftOpt, debugImg, and imageSegmentation. The code and documentation have been provided with this report.

2.1 Naive implementation

The findPeak function performs the peak searching process for a data point by first defining a spherical window at the data point of radius r, calculating the Euclidean distance between the considered point (pixel) and all other data points of the data (image pixels), and then computing the mean of the data points that lie within the defined window. After that, the function shifts the window to the mean, i.e., to a more densely populated area of the dataset, until convergence occurs, i.e., when the shift is below some threshold value of t (in our case, $t = 0.01$), reaching a peak when the data is evenly distributed in the specified window. The function returns the associated peak with the data point as a 2D numpy array.

The meanShift function calls the findPeak function for each point separately and assigns a label to each point according to its peak. Initially, all labels are initialized as -1. Peaks are compared after each call to the findpeak function and for similar peaks to be merged. Two peaks are considered to be the same if the distance between them is smaller than $r/2$. Also, if the peak of a data point is found to already exist in peaks then for simplicity its computed peak is discarded and it is given the label of the associated peak in peaks. Every time we check if the calculated peak of a data point is found to already exist in found peaks by calculating the Euclidean distance between the considered peak of the data point and the found peaks, taking the nearest peak among all the found peaks, and checking whether the nearest peak is less than $r/2$ away from the considered peak. The meanShift function returns a vector containing the label for each data point as a numpy array of size (1, a number of data points (pixels of images)) and a matrix storing the density peaks found using meanshift as its columns as a numpy array of size (a number of found peaks (a number of identified clusters), a features dimension (3D with color channels features or 5D with the color channels and x, y coordinates of each pixel)).

A new function debugImg to debug the algorithm is introduced here. This function reads the debug data, calls the meanShift function, reports the time that the algorithm takes to execute the segmentation task and the class distribution of the data points, and plots a 3D distribution of the results.

2.2 Implementation with speed-ups

It should be noted that the naive implementation of the Mean Shift algorithm is slow. Therefore, a number of modifications to speed up the algorithm and make it time-efficient are incorporated. First speed-up associates each data point that is at a distance $\leq r$ from the peak with the cluster defined by that peak. For that, a modified version of the meanShift function is introduced, where, upon finding a new peak, the Euclidean distance between the found peak and all data points is being calculated, and the points within the distance $\leq r$ from the peak are labelled with the cluster defined by that peak. The second speed-up associates the points that are within a distance of r/c (where c is a constant value) of the search path with the converged peak. For that purpose, the function findPeakOpt is introduced, which is a modification of the function findPeak that, in addition to the associated peak with a data point, outputs a vector cpts as a numpy array of the size (1, a number of data points (pixels of images)), storing a 1 for each point that is a distance of $r/4$ from the path and 0 otherwise. The meanShiftOpt function mentioned in the first speed-up calls the findPeakOpt function, finds the indexes of the data points with the value of cpts vector equals to 1, and labels those points that are on the search path with the converged peak.

As an extra speed-up, in the meanShiftOpt function, the label's array is initialized with -1 for all data points, and then the findPeakOpt function is called only for the points that are not labelled yet. Additionally, all python lists are replaced with deques which are a generalization of stacks and queues (deques support thread-safe, memory efficient appends and pops from either side of the deque with approximately the same $O(1)$ performance in either direction)[1].

[1] Source of the finding.

2.3 Implementation of segmentation

The imageSegmentation function performs the actual segmentation of an image. As an input, it takes the path to an image and the parameters that are a search window radius r, a constant value c, a feature type for the dimension of the feature vector as 3D or as 5D (specifying the color and x, y coordinates for each pixel), and a flag to indicate whether the optimized version of meanShift algorithm is used or not. After that, it reads and normalizes the image, converts it to the LAB color space, depends on a feature type, transforms or not transforms a feature vector of the image, and reshape it to have a particular structure of the input to use the Mean Shift algorithm. After the Mean Shift algorithm has been executed, the resulting segmented image is translated back into the RGB color space.

3 Experiments

3.1 Evaluation of the functions on the debugging data set

In order to check the correctness of the algorithm implementation, the implementation is checked on the clustering of the data (pts.mat which stores a set of 3D points belonging to two 3D Gaussians) provided for debugging, expecting with $r = 2$ and $c = 4$ to get two clusters.

In Figure 1, one can see that the optimized and non-optimized versions of the Mean Shift algorithm successfully cluster the debugging data and provide correct results with two peaks and the same labels assigned to corresponding pixels.

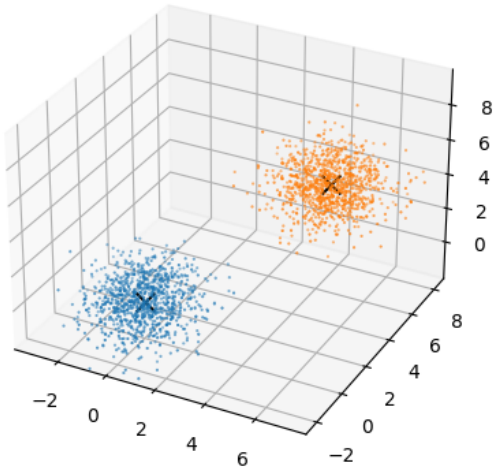
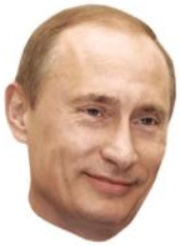


Figure 1: Clustering result of the debugging data.

	Function	Execution time (s)
1	meanShift	1.59
2	meanShiftOpt	0.32

Table 1: Comparison of the performance of the algorithm versions on the debugging data.

Additionally, an experiment was conducted to establish the effect of speeding approaches. Table 1 provides a comparison of the time efficiency of the implemented versions of the algorithm. Optimization of the Mean



(a) The first image of the size 200x200.

(b) The second image of the size 128x128.

Figure 2: Small images for experiments and evaluation.

Shift function shows a significant acceleration in the time performance, being 1.27 seconds faster. The results clearly show the importance of the speed-ups mentioned above even on a toy example.

3.2 Evaluation of the functions on small images

For experiments and evaluation, two small images presented in Figure 2 are selected in order to optimize the execution time of the algorithms and reduce the number of pixels considered for processing by the algorithm.

The images are segmented by the algorithm with different parameters:

- using non-optimized and optimized implementations of the Mean Shift algorithm.
- varying a spherical window value r in the range (2, 7, 15, 20).
- varying a constant value c in the range (2, 4, 10).
- defining a feature vector as a 3D vector specifying the color channels or as a 5D vector specifying the color channels and x, y coordinates of each pixel.

3.2.1 The first small image

The results of the experiment on the first small image with different parameters r and c with the dimension of a feature vector 3 and 5 are shown in Figures 3 and 4 respectively. In both figures, one can see that for the $r = 2$ the resulted segmented images are very close to the original image. One can easily see that the smaller the radius, the more clusters we have (the greater the number of unique peaks) and, as a result, a more detailed segmented image is obtained.

By analyzing the different values of c for a fixed r, one can easily see that the higher the value of c, the more clusters the algorithm segments. Moreover, it can be noticed that the higher the value of c, the better the visually distinguishable segments (in Figure 3 for the fixed $r = 7$ the described effect can be seen). Hence, one can conclude that increasing c significantly improves the segments.

A dramatic improvement is observed when the feature vector contains information not only about the color of the pixels, but also about the position of each pixel. In Figures 3 and 4 and more clearly in Figure 5, it can be seen that moving into 5D feature space significantly improves the segmentation. This can be explained by the fact that when using a 3D feature vector with only pixel color information, the algorithm could associate two pixels with similar RGB channel values as "similar" and assign them to the same peak, despite the fact that in terms of relative position they may be far from each other. By providing more

Figure 3: Segmentation results for various values of r and c on the first small image with a 3D feature vector.

Figure 4: Segmentation results for various values of r and c on the first small image with a 5D feature vector.

Figure 5: Segmentation results for the visual performance comparison between the optimized and non-optimized implementations.

complete information about the pixel (including the x, y coordinates of each pixel in the pixel feature vector), the algorithm can now take this information into account when determining the "similarity" of pixels that have similar color features values, but are completely far from each other in relative position in the image.

For the segmentation process with the optimized algorithm obtained with all possible combinations of r, c, and the feature vector, the time elapsed has been measured and reported in Table 2. From the results obtained, it can be concluded that the larger the radius, the faster the segmentation process. With parameter c, the opposite is true, the greater its value, the slower the algorithm performs segmentation, which is naturally based on the definition of this parameter, where this parameter reduces (divides) the radius of the considered distance in the search path. A notable case in Table 2 is when there is a combination of parameters $c = 10$, the feature vector is 5D, and the all considered range of r. Here, there is no dependence of the algorithm execution time on the radius value, a possible reason for which may be that the deceleration

of the algorithm from the maximum considered value of c non-linearly overlaps with the acceleration from an increase in the value of the parameter r, which gives difficult to explain jumps in the algorithm execution for this combination of parameters.

In order to see a time gain comparison between optimized and non-optimized implementations, an experiment with the first small image is performed with both implementations and with a fixed $c = 4$ parameter, varying r and a feature vector. The quantitative result is shown in Table 3, and the visual comparison is shown in Figure 5. Based on the results obtained, the need to use the accelerated version is once again clearly proven.

Table 2: Time gain comparison for various values of r and c with the optimized implementation for the first small image.

	Execution time (s)					
	3D			5D		
r/c	2	4	10	2	4	10
2	17.39	29.43	29.43	66.29	60.71	60.18
7	4.56	15.71	15.71	24.40	51.43	82.39
15	0.93	3.88	3.88	13.15	39.21	148.91
20	0.39	1.39	1.39	10.24	37.90	139.99

Table 3: Time gain comparison between optimized and non-optimized implementations ($c = 4$ is fixed) for the first small image.

	Execution time (s)			
	Non-optimized		Optimized	
r	3D	5D	3D	5D
2	199.00	272.49	29.43	60.71
7	380.05	291.91	15.71	51.43
15	442.05	568.87	3.88	39.21
20	454.29	902.64	1.39	37.90

3.2.2 The second small image

The similar experiments are performed on the second small image to confirm the drawn conclusions and verify the observations made on the first small image. Experiments to compare the time gain performance of the optimized and non-optimized versions of the implementations are not conducted due to the fact that a significant dominance of the optimized version is proven by the experiments with the debugging data and the first small image.

The results of the experiment on the second small image with different parameters r and c with the dimension of a feature vector 3 and 5 are shown in Figures 6 and 7 respectively. A time gain comparison experiment is shown in Table 4. The results obtained confirm the conclusions made in the first experiment.

Table 4: Time gain comparison for various values of r and c with the optimized implementation for the second small image.

	Execution time (s)					
	3D			5D		
r/c	2	4	10	2	4	10
2	3.55	4.31	5.30	13.95	15.34	15.92
7	2.22	3.95	17.58	6.08	10.28	16.35
15	0.28	1.27	8.17	3.69	10.21	24.34
20	0.20	1.16	7.49	2.97	10.19	36.96

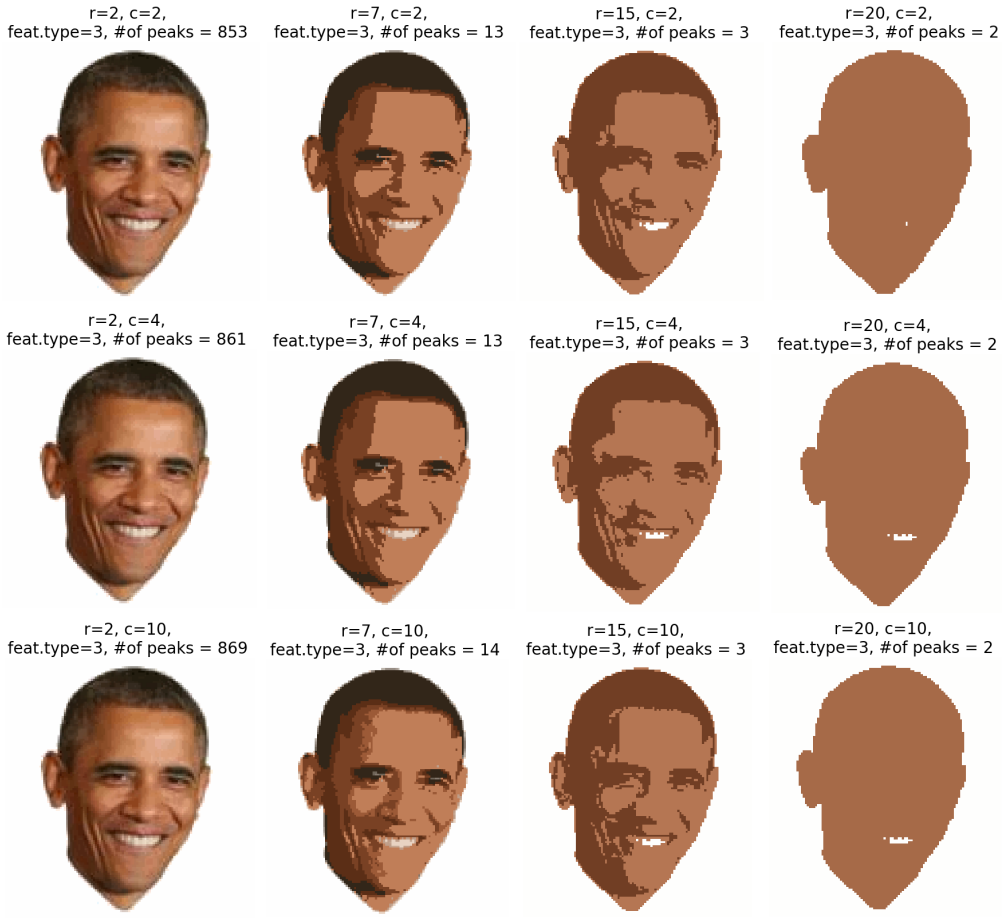


Figure 6: Segmentation results for various values of r and c on the second small image with a 3D feature vector.

3.3 Evaluation of the functions on the validation images

The next round of experiments is conducted on the images from the Berkeley Segmentation Dataset. In this experiment, the constant parameter c does not vary in order to reduce the execution time of the algorithm, since the effect that this parameter has on image segmentation has been established. The following parameter combinations are used:

- varying a spherical window value r in the range (4, 10, 15, 25).
- a constant value $c = 4$ is fixed.
- defining a feature vector as a 3D vector specifying the color channels or as a 5D vector specifying the color channels and x, y coordinates of each pixel.

In this part, there is also no experiment with comparing the optimized and non-optimized implementations, since the effect was established in previous experiments and on these large-size images, the non-optimized algorithm takes a huge amount of time to be executed.

The results of the segmentation of the first image with the 3D feature and 5D feature spaces are shown in Figure 8 and 9 respectively. The results obtained confirm the conclusions and observations made in the experiment with small images. You can clearly see how as the radius increases, the number of clusters found decreases and the segmented image becomes less clear. Also, comparing the results of 3D and 5D feature spaces, one can see how the algorithm is more successful in dealing with segmentation, having information about the relative positions of pixels in addition to the color features.

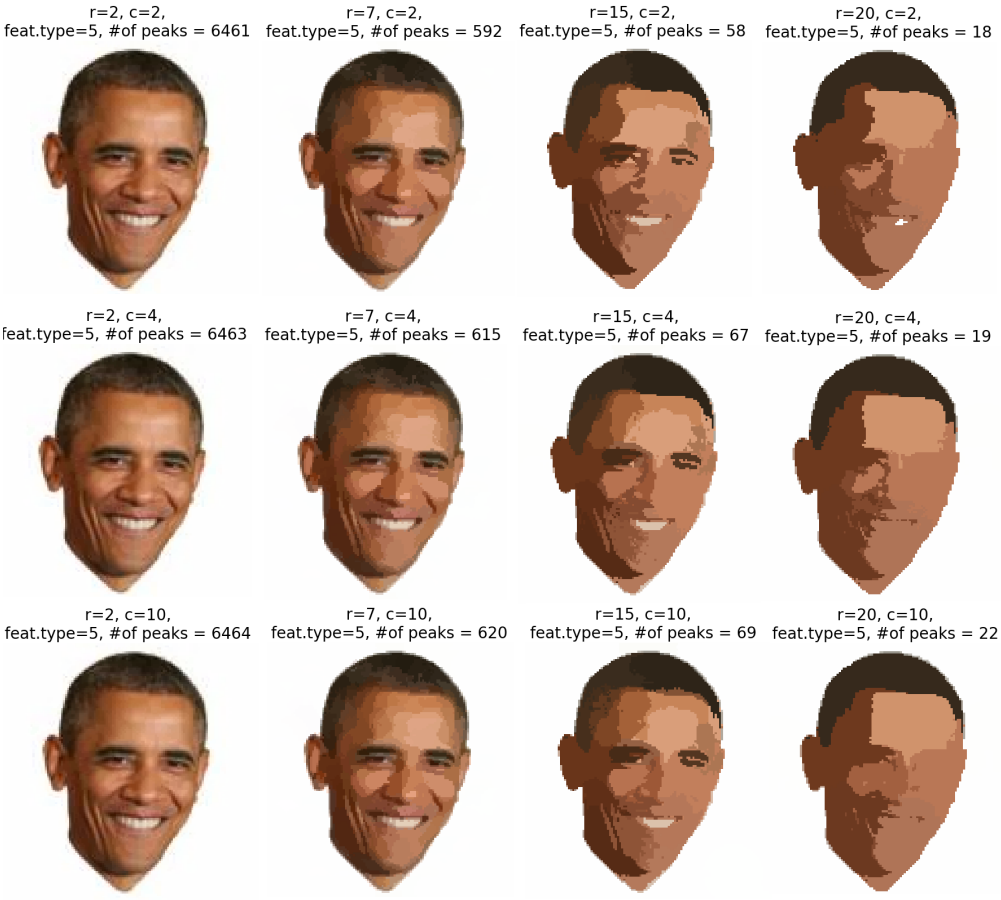


Figure 7: Segmentation results for various values of r and c on the second small image with a 5D feature vector (the segmented image is on top, and its corresponding 3D cloud-point cluster is on the bottom).

In Figure 8 and 9, one can also see under each segmented result visualization of pixel clustering in the 3D cloud point plot, which shows how the number of clusters changes when the r parameter changes. It can be seen that the larger the radius, the fewer clusters we have, which is reflected in the color variations change on the graphs.

The segmentation results for various parameter combinations of the second image are shown in Figure 10 and 11. The results for the third image are show in Figure 12 and 13. All findings follow the same pattern as described above for the first image.

3.4 Gaussian and median filtering

Following a suggestion to propose additional processing steps in order to improve the segmentation results, filtering techniques are introduced as the preprocessing steps before applying the Mean Shift algorithm. Gaussian and Median filtering techniques [2] are compared. The intuition behind that preprocessing step is that noise might influence segmentation boundaries identification of the algorithm. The Gaussian filtering is highly effective in removing Gaussian noise from the image and enhancing image structure. The Median filtering is highly effective in removing salt-and-pepper noise. This experiment is performed with the third image (Figure 14). To do this, we fix the parameters $r = 25$, $c = 4$, and the 5D feature space.

The Figure 15 shows the results of the experiment. One can notice that preprocessing in both cases slightly changes the color palette of the image. Visually, it can be seen how segmentation successfully copes with processing images. The figure with the result of segmentation of the Gaussian image shows that there are fewer peaks found, which may be caused by a smoother input image after filtering and some edges were

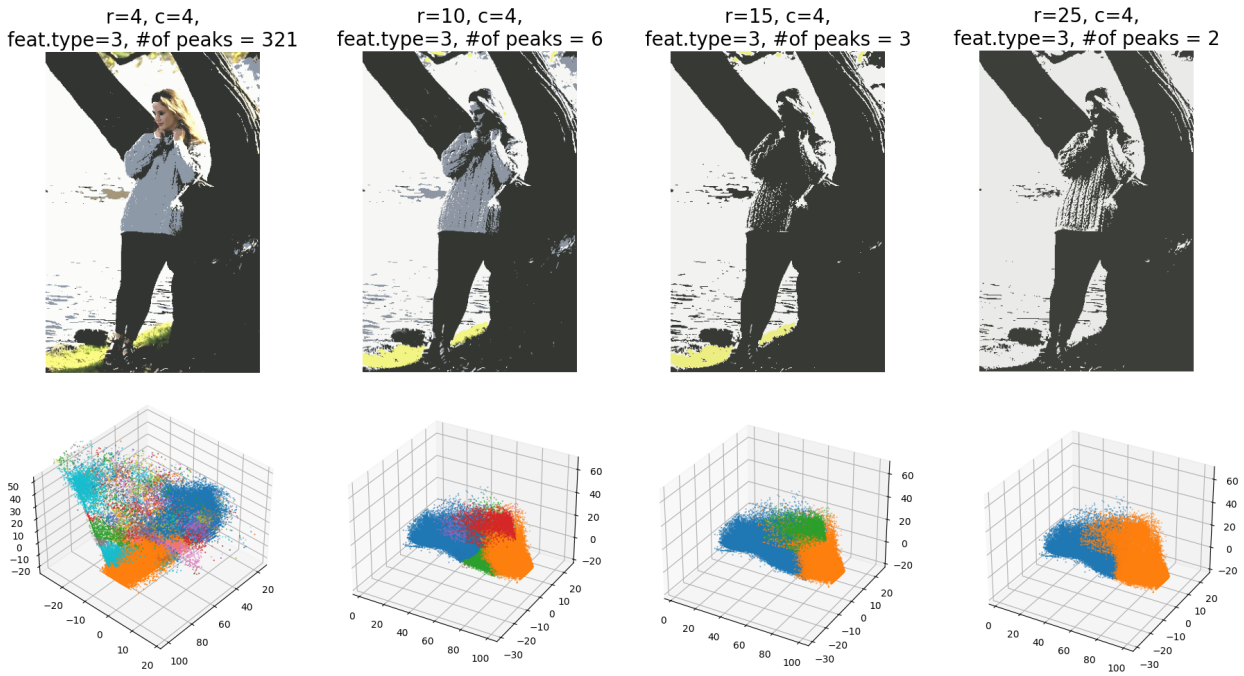


Figure 8: Segmentation results for image 1 with a 3D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

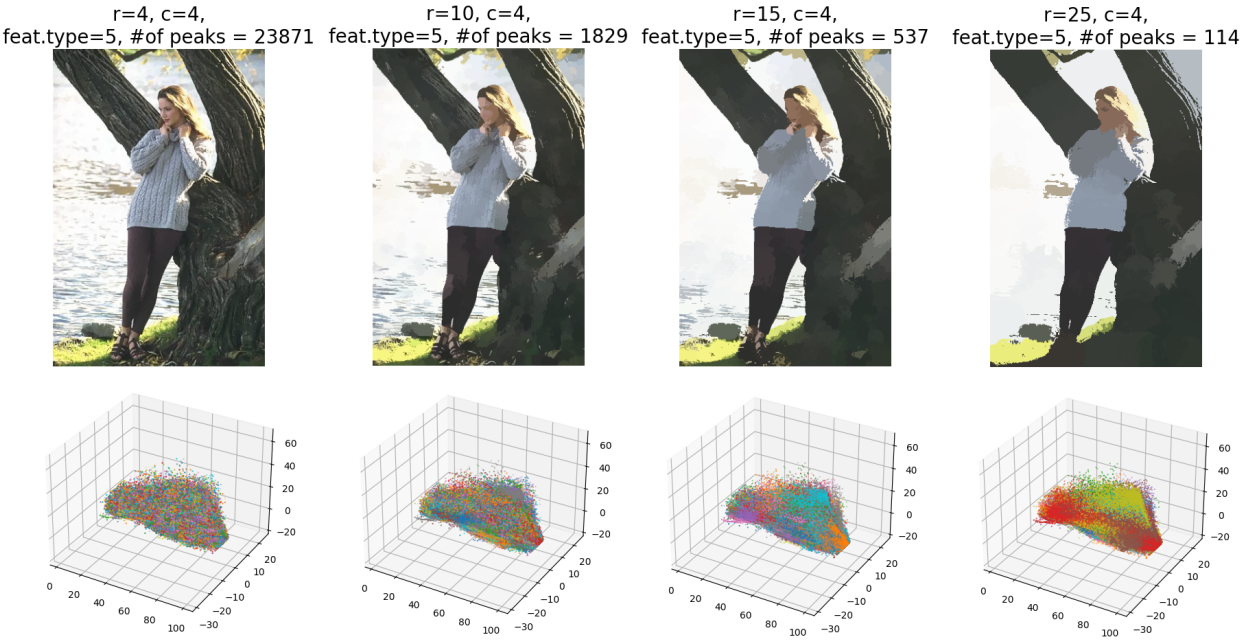


Figure 9: Segmentation results for image 1 with a 5D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

blurred.

The Table 5 shows the results in the time gain of the optimized algorithm on fixed parameters with the proposed preprocessing steps. It is worth noting that the preprocessing significantly speeds up the

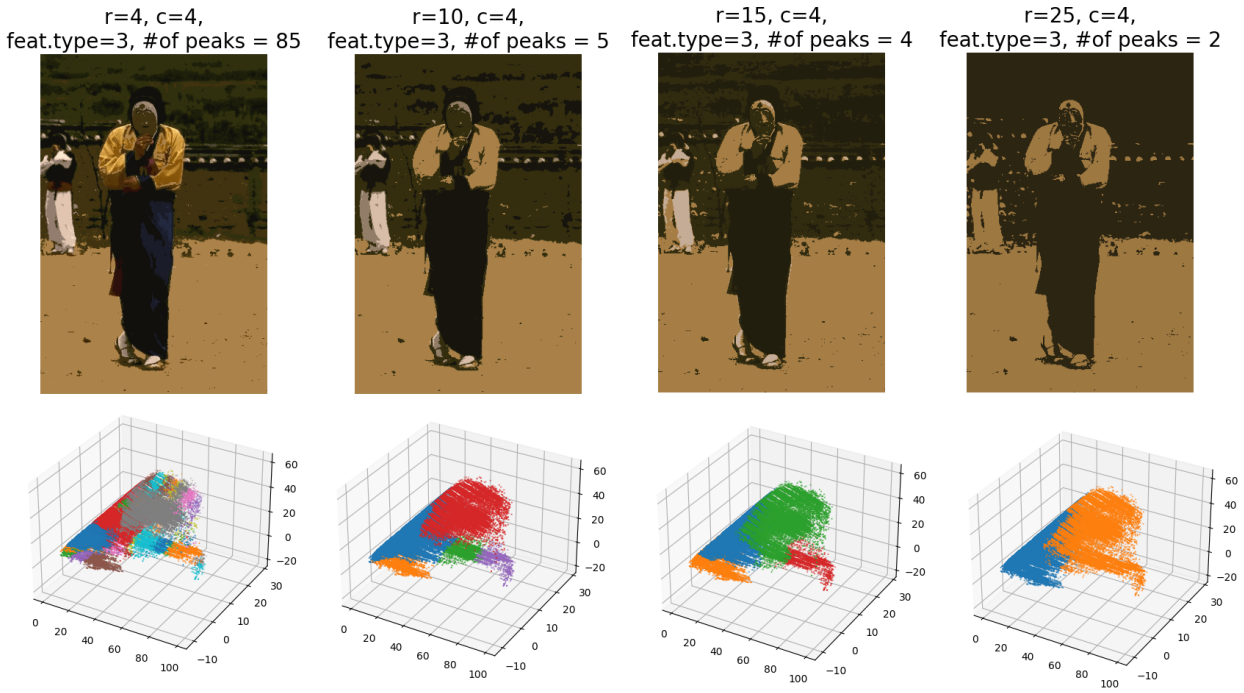


Figure 10: Segmentation results for image 2 with a 3D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

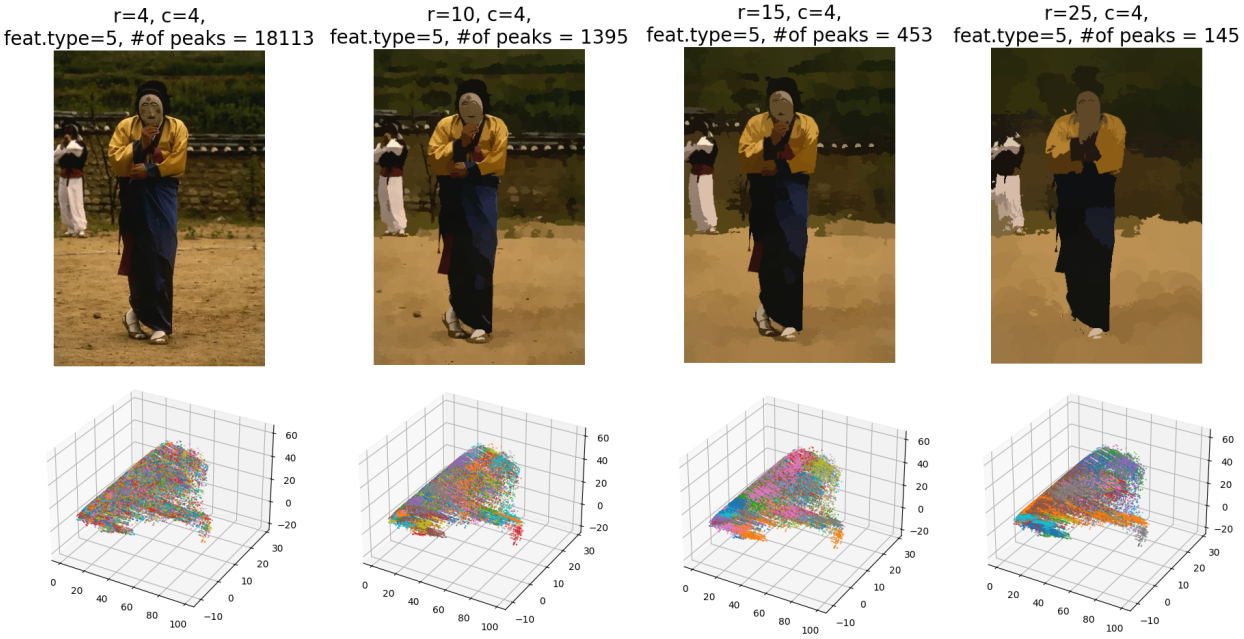


Figure 11: Segmentation results for image 2 with a 5D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

segmentation process.

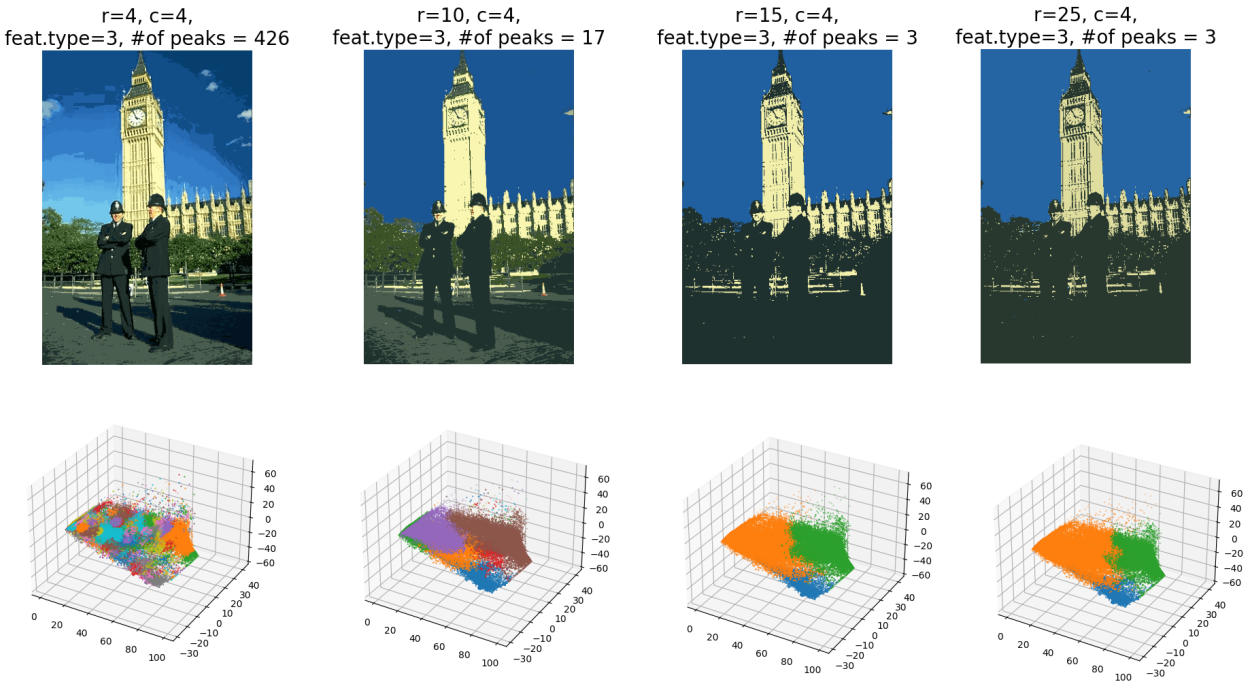


Figure 12: Segmentation results for image 3 with a 3D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

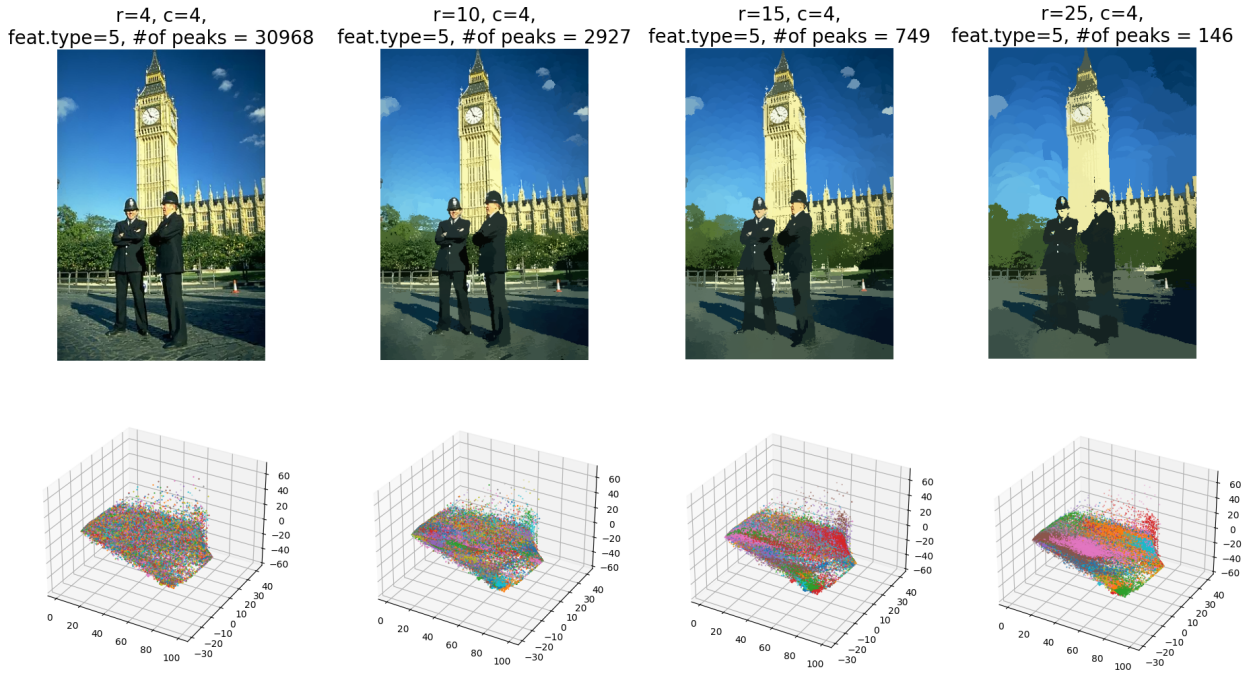


Figure 13: Segmentation results for image 3 with a 5D feature vector (the segmented image is on top, and its corresponding cloud-point cluster is on the bottom).

	Image	Execution time (s)
1	Original image	1417.43
2	Gaussian filtered image	689.82
3	Median filtered image	909.51

Table 5: Comparison of the performance of the optimized algorithm on different preprocessed images.

Figure 14: Images comparison after applying filtering techniques.

Figure 15: Comparison of segmentation results for different filtering techniques.

4 Conclusions

In this work, the Mean Shift algorithm is implemented and optimized. The influence of parameters on the segmentation process by the algorithm was established. The experiments show that this algorithm successfully copes with the task of image segmentation. Despite the fact that all possible speedups of the algorithm were implemented in the Python programming language, including acceleration by the numba software package, since it was found that 60% of the execution time is occupied by the cdist function, for optimization and parallelization of a similar function np.linalg.norm, which did not show the claimed improvement and therefore the cdist function is used in the final implementation, work on further optimization of the algorithm on large

images remains future work.

References

[1] Miguel Á. Carreira-Perpiñán. "A review of mean-shift algorithms for clustering". In: *CoRR* abs/1503.00687 (2015). arXiv: 1503.00687. URL: http://arxiv.org/abs/1503.00687.

[2] Arvind Kumar and Sartaj Singh Sodhi. "Comparative Analysis of Gaussian Filter, Median Filter and Denoise Autoenocoder". In: *2020 7th International Conference on Computing for Sustainable Global Development (INDIACom)*. 2020, pp. 45–51. DOI: 10.23919/INDIACom49435.2020.9083712.