Maastricht University

Computer Vision

Assignment 1

Abel de Wit
(i6139815)

May 17, 2021

1 Introduction

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image onto multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. [1]

2 Methodology

In this project, the goal is to utilize the Mean-Shift algorithm [2] to create clusters from an input image. This is achieved by representing each pixel in the image as a vector in either three- or five dimensional space. With this representation, pixels that have similar values in all dimensions will be positioned closer to each-other. Using this fact, clusters of pixels can be made. The center point of a cluster is found by iteratively shifting the search window towards higher densities in the multi dimensional representations. The iteration is stopped whenever the shift of the search window is smaller than some value, in our case (0.01).

To apply this algorithm, there are three steps to take:

2.1 Pre-processing

When the program receives an image as input the image is represented as a numerical matrix with dimensions (width x height x color_space). Most images are represented in the RGB color space, but for segmentation the CIELAB colorspace is a better representation of visible changes in color, hence we convert the image to this color space using the *'scikit-image'* package. Another step in the preprocessing is scaling and blurring of the image. Blurring the image allows for better segmentation, and scaling the image allows for faster processing as there are less pixels to handle.

The last (optional) step is creating an embedding for the position of the pixels, allowing for better clustering not only based on color but on position in the image as well. This embedding is achieved by added two aditional layers to the image, containing each pixel's x and y coordinate in the image in a seperate layer. The result is an image with five dimensional features.

2.2 Finding peaks

When the pre-processing is done, the next step is to find the peaks of our image. This is achieved by flattening the image, resulting in an array of vectors. For each vector in this array, the algorithm tries to find a peak based on the Mean-Shift technique. The technique consists of finding the nearest neighbors to the current pixel/data point, calculating the mean of these points, and shifting the search window to this new mean. When the shift does not yield a new mean, the found peak is returned.

2.3 Segmenting image

When each pixel has been handled and a peak is known for each, the pixels that share a peak (or ones that are very similar), will receive the same label. The 'mean' color of this segment is the values of the peak that all the pixels in a cluster share. With these labels and cluster peaks, a new image can be constructed, assigning the cluster color to each pixel in the image that has that respective label. The result is an image that is segmented.

2.4 Optimizations

Part of the assignment was improving the Naive Mean-Shift algorithm that processes each pixel separately. Because each pixel had to be visited and for each of those a peak has to be found, large images are close to impossible to process because of the performance bottleneck. There are however two assumptions that can be made which will speed up the algorithm significantly.

2.4.1 Basin of attraction

Through thee way the mean-shift algorithm works, it is safe to assume that the first time a mean is calculated based on the neighbors of the current pixel, most of these neighboring pixels will essentially converge to the same peak. By utilizing this assumption, a list of all the neighboring pixels can be saved, and when the current pixel's peak has been found, all of these initial neighbors can safely get the same label.

2.4.2 Search path

While shifting through the data points, continuously calculating the point densities and moving to the highest result, the search window travels along a 'path' throughout our data. Because each time the highest density is taken as the new mean, the second assumption that we can make is that any point that is close to the 'path' of our mean will very likely also travel along the same path or one very similar, ending up at the same peak. Hence, we can also remember the points that we encountered and safely label those with the same peak as well.

3 Experiments & Results

3.1 Optimization improvements

By implementing the optimizations, a large portion of pixels can be skipped in the algorithm. But how much does it improve the performance? To test this, the algorithm was tested on the *Berkeley Segmentation Dataset, #368078* [1].

This image has dimensions (321×481), resulting in 154.401 pixels. For this experiment, the image was scaled by a factor of 0.5 in order to make the non-optimized run-time manageable. There is no positional encoding in this test and So on an image of 38400 pixels the resulting run-times can be seen in Table 1.

	Total runtime	**Average iterations /s**
Non-optimized	05:38	113.51 it/s
Optimized	00:02	16422.83 it/s

Table 1: Run-time results

The improvement in time between the two implementations is 169 fold, and the amount of iterations per second increased by a factor of ≈ 145. From this we can conclude that these improvements are not only beneficial but almost a necessity when working with large images. The resolution of images is increasing while the need of processing them is as well, run-times of over 5 minutes for a 'small' image are not a possibility anymore.

[1] `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images/color/368078.html`

3.2 Parameter tuning

The Mean-Shift algorithm has quite some parameters to tune, especially when taking the optimizations into account.

- r : The size of the search window
- c : The size of the search-path window
- *feature_type* : Positional encoding

In order to find the best parameters, a small image of a landscape (Figure 1) and another of a frontal face was used. A grid search was performed with the parameters specified above, testing several possibilities of combinations and saving their results and run-times.

Parameter	**Tested Values**
r	[2, 10, 20, 50]
c	[2, 4, 10]
scale	[0.25, 0.5, 1]
feature	[3D, 5D]

Table 2: Tested parameter values

The r parameter, or search window, indicates the distance at which neighbors are considered for the Mean-Shift algorithm. The larger this search window, the more data-points will be taken into account resulting in more overlapping peaks and hence less segments. This intuition is confirmed in both Figure 2 and 3, where a search window of size 2 has many small segments, while the (maybe too large) window of 50 produces only two or three segments for the whole image. The influence of r on performance is also evident, both figures show that by increasing the search window and hence the basin of attraction, computation time is decreased significantly. It is therefore important to make a well balanced choice on how detailed the segments should be.

The c parameter, or search-path window, indicates which points that lie around the search-path towards a peak are considered to be part of the same cluster. Because this distance is not c directly, but rather r/c, this parameter will have an inverse effect when increased. If c is small, a large portion of the initial search window will also be part of the search path, while a large c decreases

this effect. The influence of this parameter can be seen best in the clouds of Figure 4. When this parameter is smaller, there is less *'color bleeding'* as more data-points are involved in the search path of early peak searches. As the search path gets more narrow, less data-points will be part of initial searches and get more accurate clusters or even their own.

The *feature_type* parameter involves the encoding of pixel position in their vectors. This allows the algorithm to better distinguish not only color but also segment positions. The difference between having this encoding and not can be seen very clearly between Figure 2 and 3. With the same window size, the images that have the position embedded show far greater detail and better segments. This improvement does come at a cost however, as two extra dimensions to calculate mean less performance. The 3D encoded image with window size 50 needed only 0.0973 seconds to come up with two segments, while the 5D encoded image needed 8.5315 seconds for the same window size. This is once again, a difficult deliberation between performance and accuracy.

The same experiments on the face can be found in the experiments folder included with the source code.

3.3 Berkeley results

In the Berkeley test images we can really test how well the segmentation performs. In Figure 6 to 17 the results of applying the Mean-Shift with different parameters can be seen. With test image #181091, it is clear that the grey sweater poses a problem against the white/grey-ish background, but the algorithm seems to deal with this very well, probably because of the spatial encoding as well. In the 3D plots of the clusters it also becomes evident what influence the search window has on the final clusters. With a search window of 30 (Figure 11), there are large clumps of the same color while with a search window of 5 (Figure 9) a gradient can be distinguished between the many clusters.

Another thing that came up with the experiments was the influence of a different implementation of the optimizations. While this report's implementation checks if a label has been assigned to pixels that fall within the basin of attraction and search-path and only labeling them if they have not, a discussion with a peer brought up that in their case they always override the label of every pixel within those search regions. When comparing results this gave an interesting difference. In the Berkeley image #368078 segmentation, there is a gradient in the sky. With a search window of 30, this results in three layers of blue 8. With this reports implementation, as can be seen in the figure, the darker blues 'overlap' the lighter ones because the algorithm moves from the top-left to the bottom-right. But when pixels that fall within the search regions are overwritten altough they do have a label, the gradient points 'upwards' where the lighter colors 'overlap' the darker ones. This was an interesting find between two different implementations.

4 Conclusion

The Mean-Shift algorithm seems to be very suited for the task of image segmentation. While similar colors might create confusing segmentations if only using colors as features, when positional information is encoded in the feature vectors the Mean-Shift is able to distinguish most elements and small changes in pictures.

The algorithm does however have it's pitfalls. There are still many components that are computationally heavy, and the algorithm doesn't seem to be suited for either very large pictures (4K) or video segmentation, based on the difficulties encountered during this assignment. The idea however, of representing objects and other things in an image based on their color profile (and position) seem like a good fit for simple image segmentation such as this assignment.

References

[1] L. G. Shapiro and G. C. Stockman, *Computer vision*. Prentice Hall, 2001.

[2] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, 1995.

Appendix

Figure 1: Image used for parameter tuning

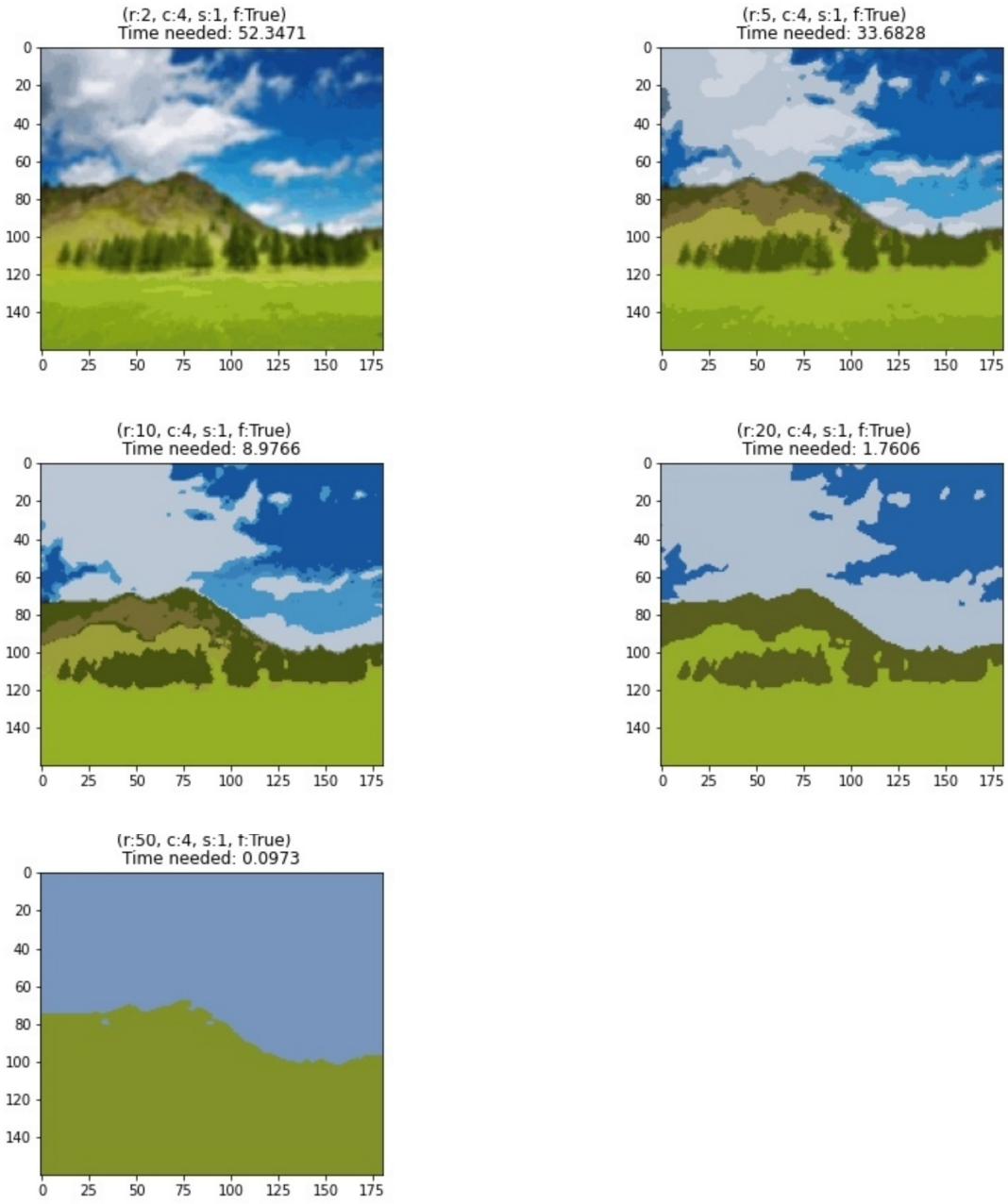

Figure 2: Different window sizes (3D features)

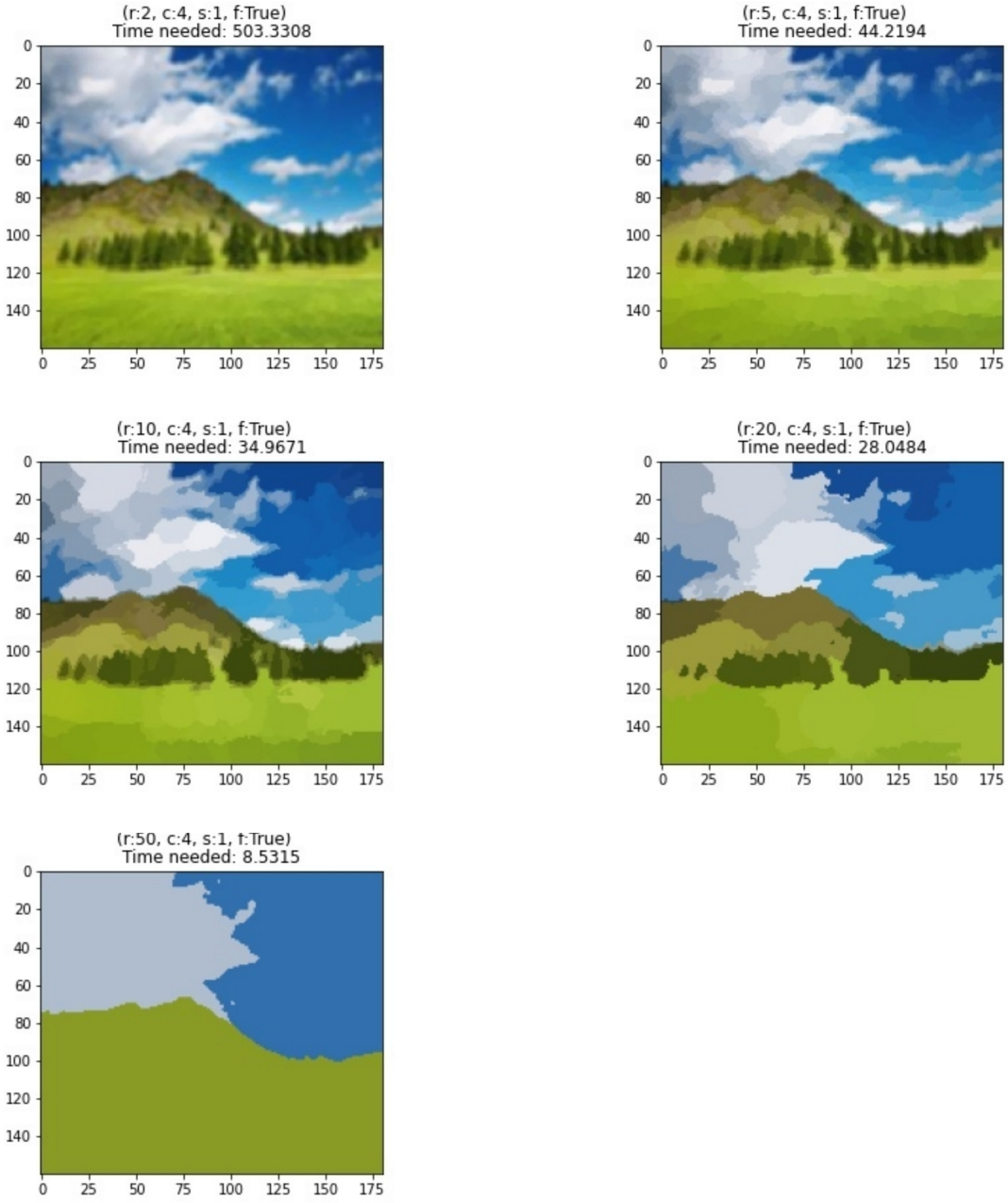

Figure 3: Different window sizes (5D features)

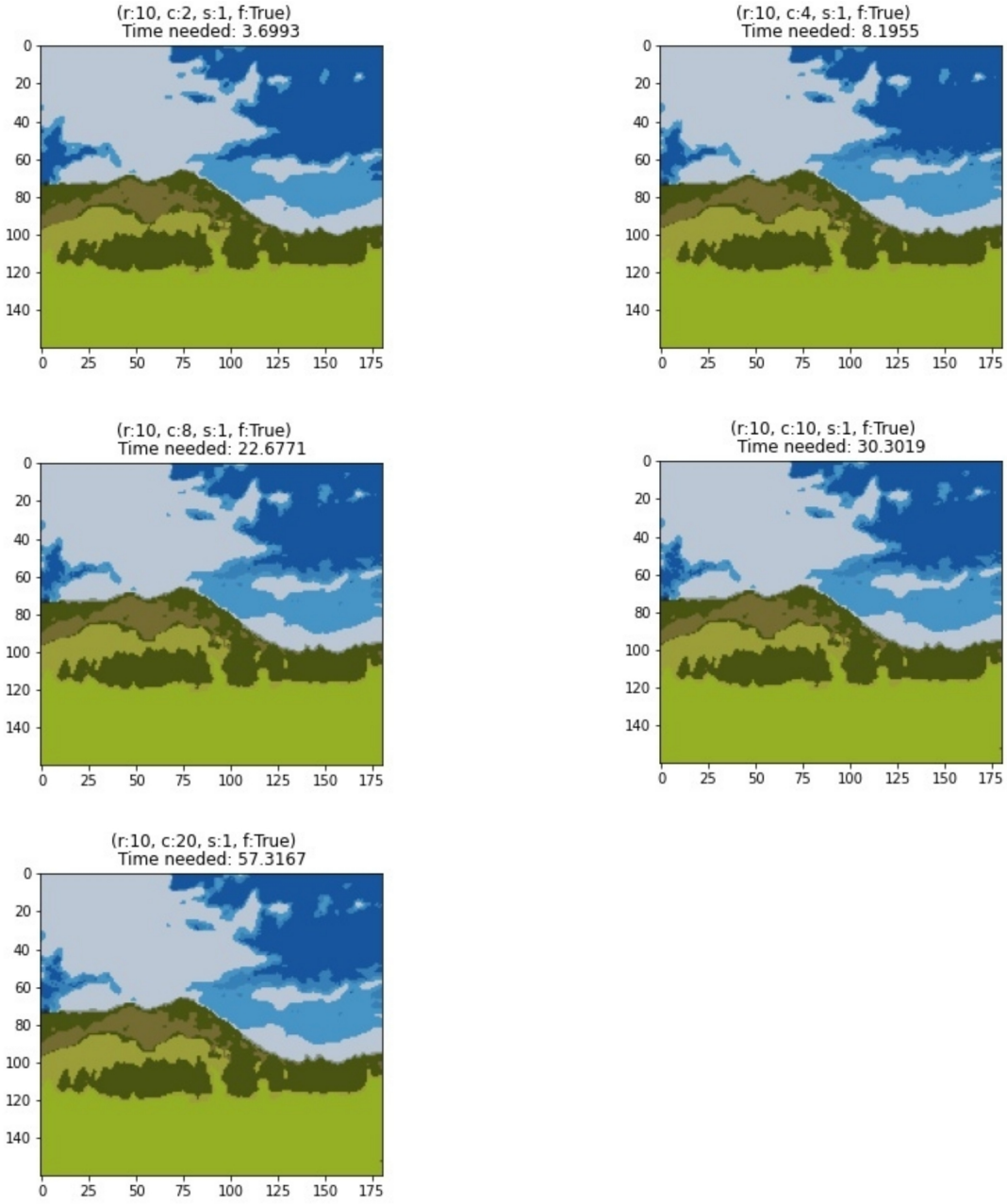

Figure 4: Different search path sizes (3D features)

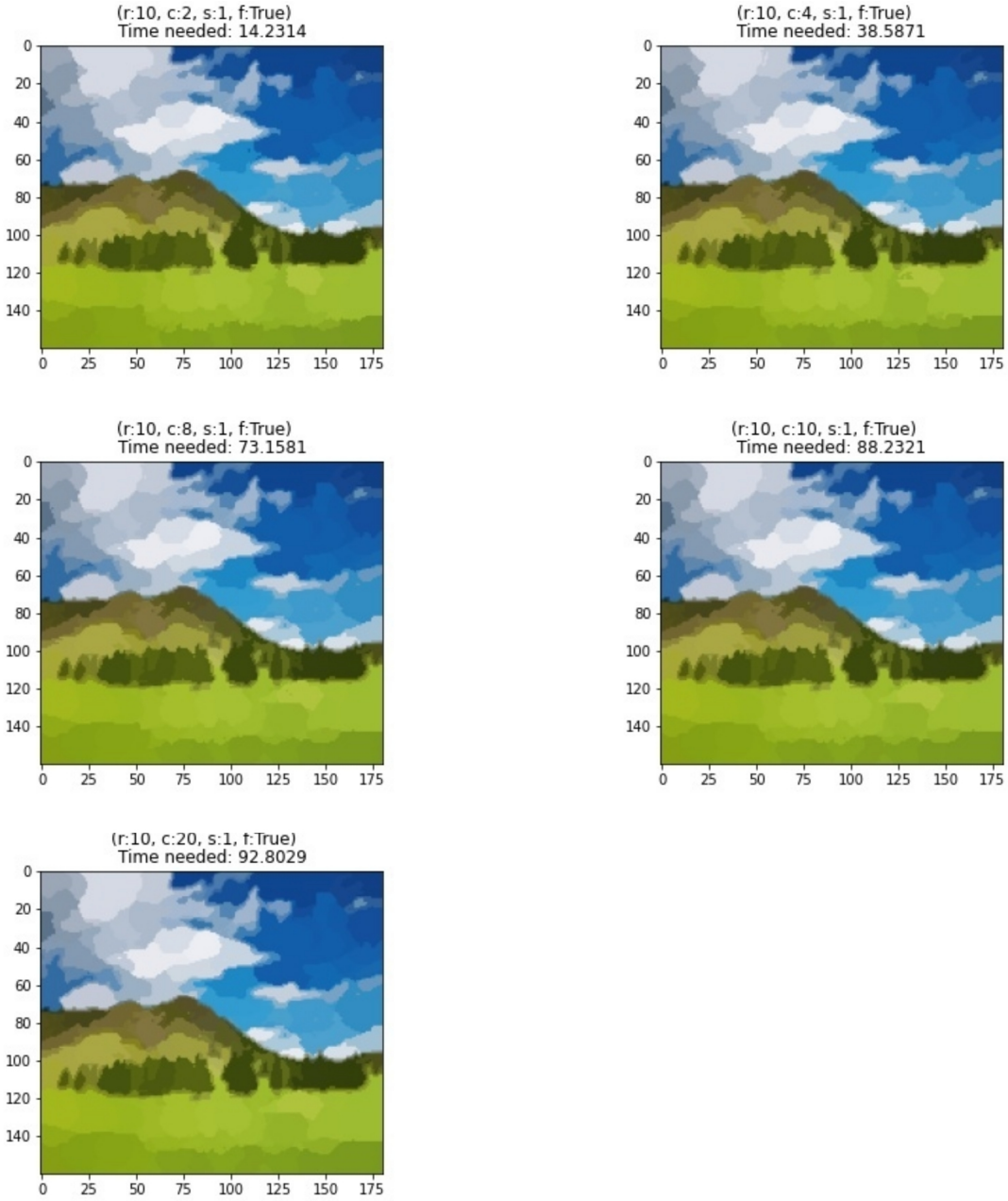

Figure 5: Different search path sizes (5D features)

Test results of Berkeley #181091

Figure 6: r=5, s=0.5, feature_type=5D

Figure 7: r=20, s=0.5, feature_type=5D

Figure 8: r=30, s=1, feature_type=5D

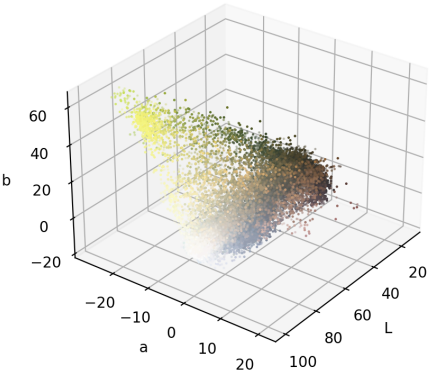

Figure 9: r=5, s=0.5, feature_type=5D

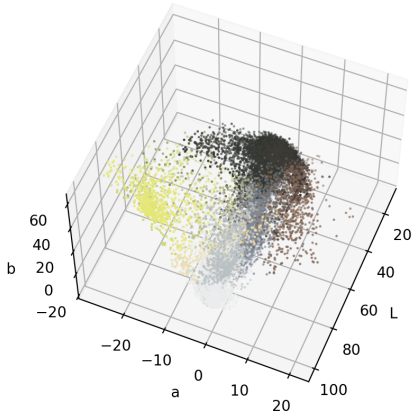

Figure 10: r=20, s=0.5, feature_type=5D

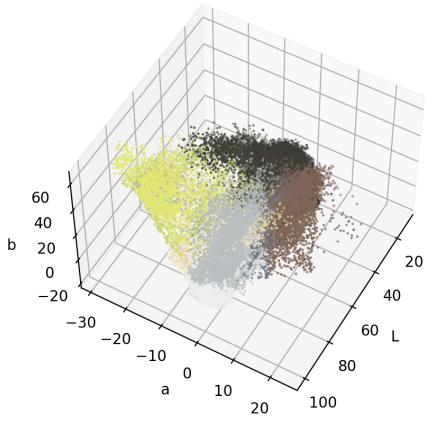

Figure 11: r=30, s=1, feature_type=5D

Test results of Berkeley #368078

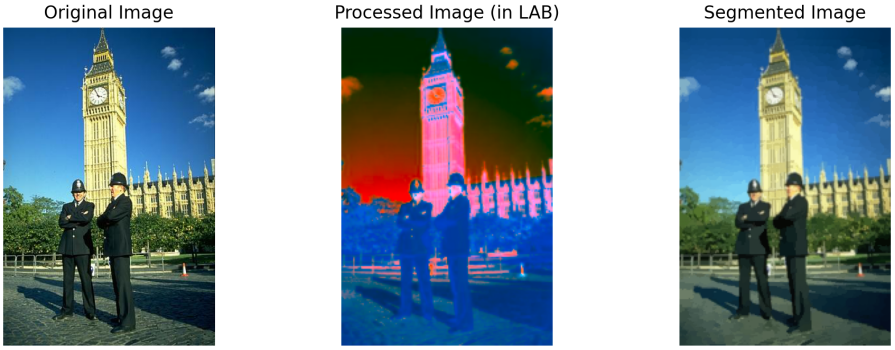

Figure 12: r=5, s=0.5, feature_type=5D

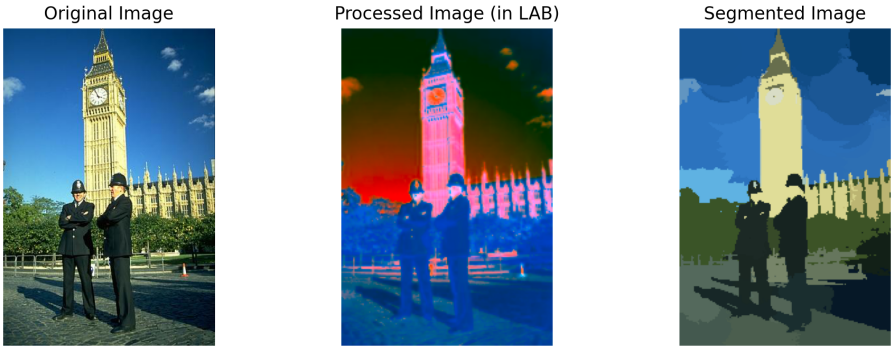

Figure 13: r=20, s=0.5, feature_type=5D

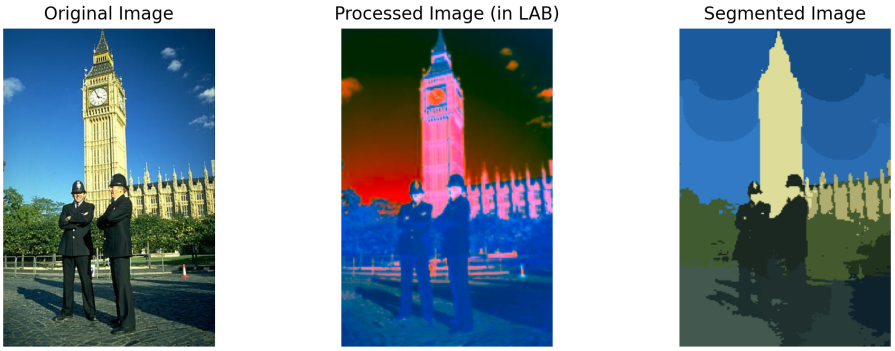

Figure 14: r=30, s=0.5, feature_type=5D

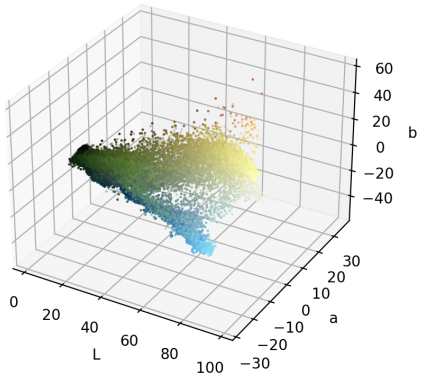

Figure 15: r=5, s=0.5, feature_type=5D

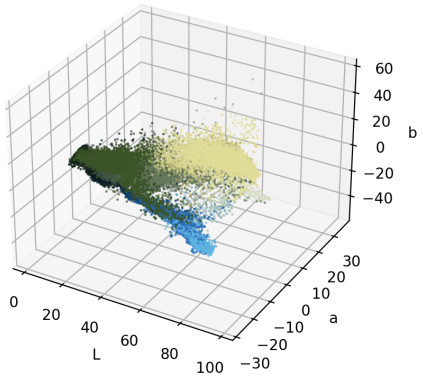

Figure 16: r=20, s=0.5, feature_type=5D

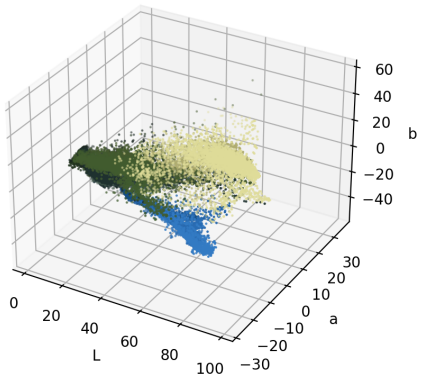

Figure 17: r=30, s=0.5, feature_type=5D