EE475 Homework #5

Batuhan Tosun, ***2017401141***

I. NOISY EDGE DETECTION

In this question, we work on the Circles image, which is centered at (101, 101) and the radii are 20, 40, 60, and 80 pixels. Color the concentric circles with gray levels of 64, 128, 192 and 255. So, we know the exact edge locations.

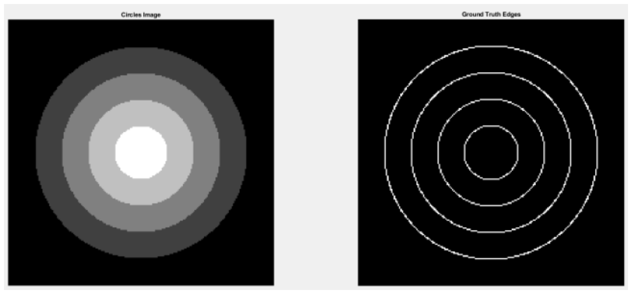

Fig 1. The Circles Image and the Ground Truth for the Edges

A. Performances of the Different Edge Detectors on Clean Image

In this part of the question, there is no noise on the image. We compare the performances of the Sobel, LoG and Canny edge detectors. For Sobel and Canny edge detectors MATLAB's edge function is used with 'Sobel' and 'Canny' parameters. For LoG edge detector, I implement my own function. For, LoG edge detection, first the image is filtered with a Gaussian filter and then convolved with the given 3x3 kernel. Finally, the resulting image is gone through the strong zero crossings testing in all-four directions find the edges.

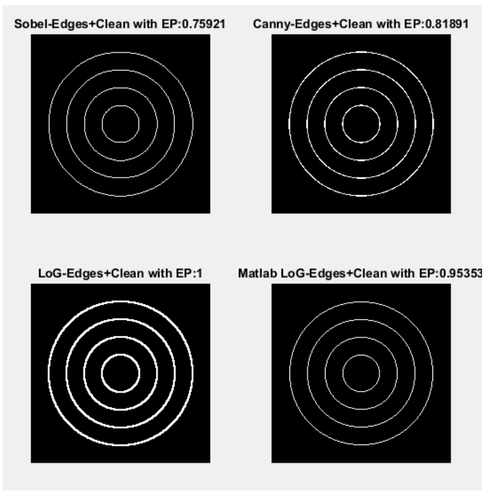
Sobel-Edges+Clean with EP:0.75921
Canny-Edges+Clean with EP:0.81891
LoG-Edges+Clean with EP:1
Matlab LoG-Edges+Clean with EP:0.95353

Fig 2. The results of Sobel, Canny and LoG (my and Matlab's) Edge Detectors on Clean Image

It can be observed that the performance (EP score) of the LoG edge detector is better than the others. Also, we can observe that the Canny gives a better result than Sobel. Even though the EP score of the LoG edge detector is better, this might not be directly related to its performance but rather the thickness of the resulting edges. If a thinning is algorithm is applied on LoG edge detector, its EP would be close to others.

B. Performances of the Different Edge Detectors on Noisy Image

In this part of the question, Gaussian noise N(0, 484) is added to the image. Therefore, we compare the performances of the Sobel, LoG and Canny edge detectors in the presence of noise.

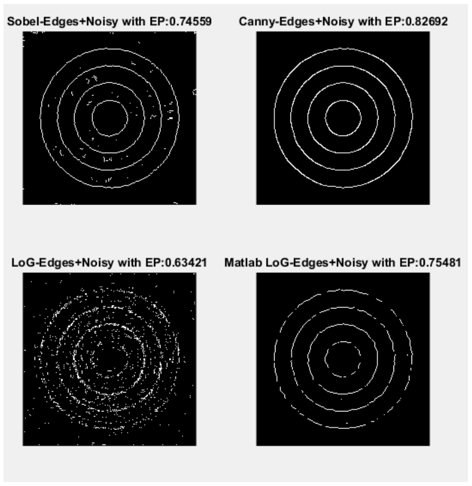
Sobel-Edges+Noisy with EP:0.74559
Canny-Edges+Noisy with EP:0.82692
LoG-Edges+Noisy with EP:0.63421
Matlab LoG-Edges+Noisy with EP:0.75481

Fig 3. The results of Sobel, Canny and LoG (my and Matlab's) Edge Detectors on Noisy Image

It can be observed that the performance (EP score) of the Canny edge detector is better than the others. There is a slight difference between the EP scores of MATLAB's Sobel an LoG edge detectors. However, by looking at the resulting images, we can say that the performance of MATLAB's LoG is better than Sobel in the presence of noise. Also, the success of Canny edge detector comes from its multi-stage thresholding mechanism.

	Sobel	**Canny**	**myLoG**	**LoG**
Clean	**0.7592**	**0.8189**	**1.0000**	**0.9535**
Noisy	**0.7456**	**0.8269**	**0.6342**	**0.7548**

Tab 1. The EP results of Sobel, Canny and LoG (my and Matlab's) Edge Detectors on Noisy Image

II. EDGE LINKING AND COMPLETION

A. *Finding the Gradient Field and Obtaining Its Histogram*

In this part of the question, the Amasya image is filtered with a small Gaussain filter (with $\sigma = 1$).

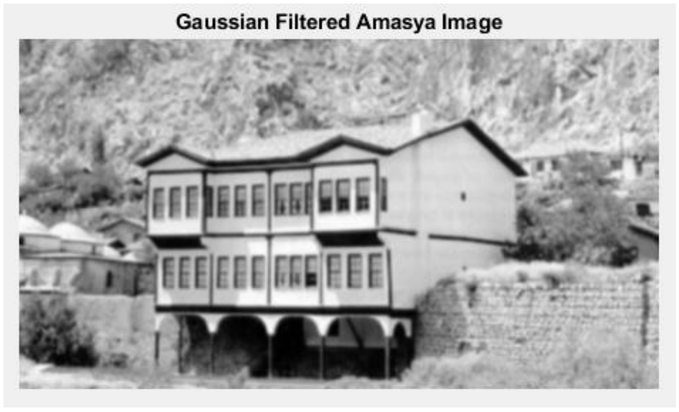

Fig 4. Gaussian Filtered Amasya Image

Then, its gradient field is found. The magnitude of the gradient field is given below:

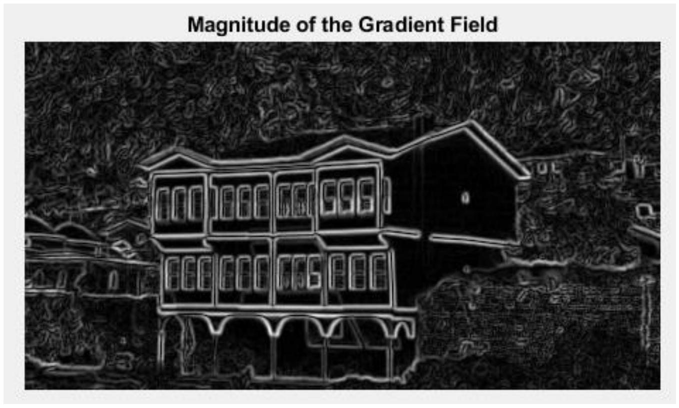

Fig 5. Magnitude of the Gradient Field of Amasya Image

The histogram of the gradient field is obtained, and the 85-percentile point is found for high-thresholding.

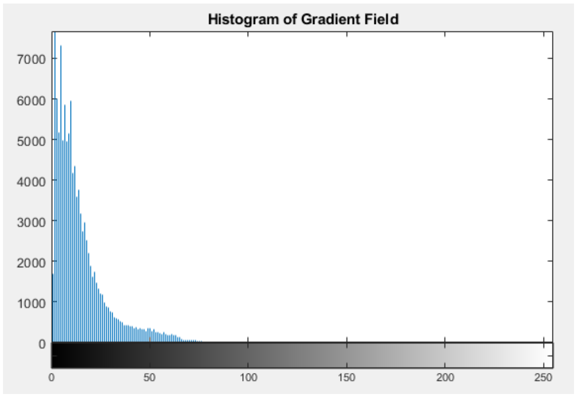

Fig 6. Histogram of the Magnitude of the Gradient Field

As a result, the high threshold value is found as 27 and the low threshold value is 13.5 (due to 2:1 relationship).

B. *Finding Edge Candidates using the High-Threshold*

In this part of the question, the strong edge candidates are found using the high-threshold. The resulting strong edges are given below:

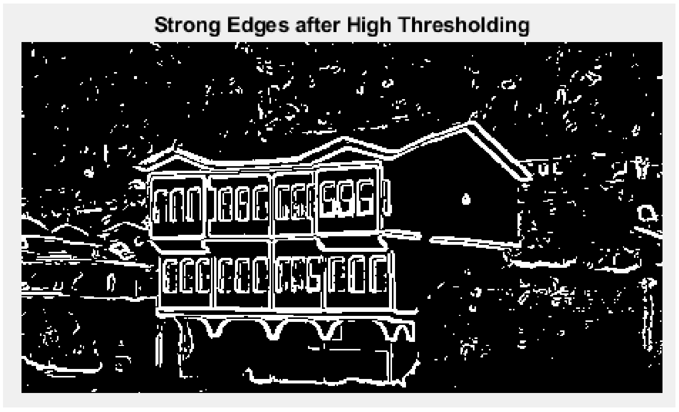

Fig 7. Strong Edges after High Thresholding

C. *Applying Hysteresis Thresholding to Connect Edges*

In this part, low-thresholding output is used to continue the strong edges.

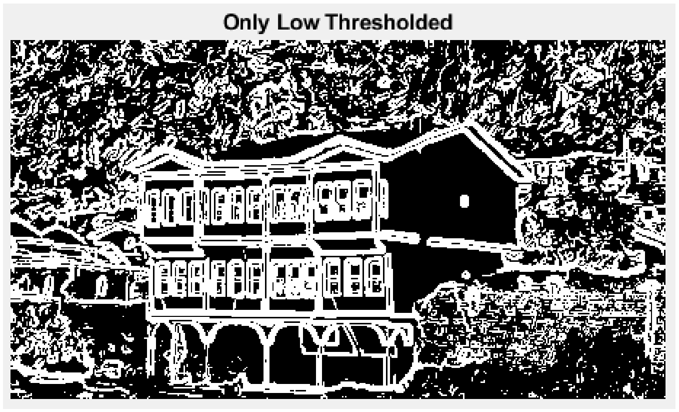

Fig 8. Weak+Strong Edges after Low-thresholding

Therefore, hysteresis thresholding is applied to connect the edges. The edges in the high-threshold image is continued if there are overlapping or 8-neighbor edges in the low-threshold image. All other low-threshold edge candidates are discarded. The resulting image is given below:

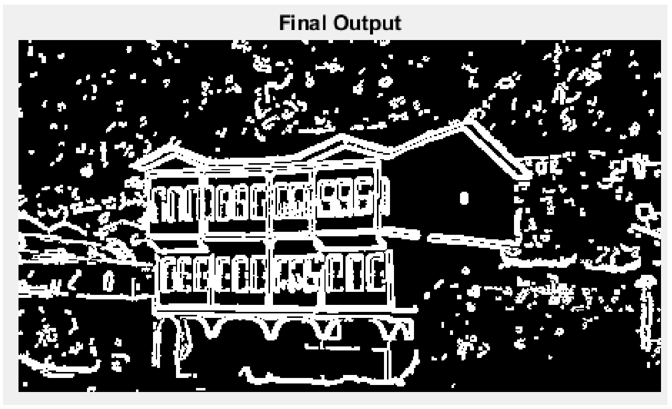

Fig 9. The Resulting Edges after Hysteresis Thresholding

It should be noted that the resulting edges are thick. One could use one of the thinning algorithms such as **"erosion"** operation to overcome this problem.

III. SEGMENTATION BY REGION GROWING

In this question, the aim is to accomplish a segmentation task on Berkeley_deer image using Seeded Region Growing algorithm.

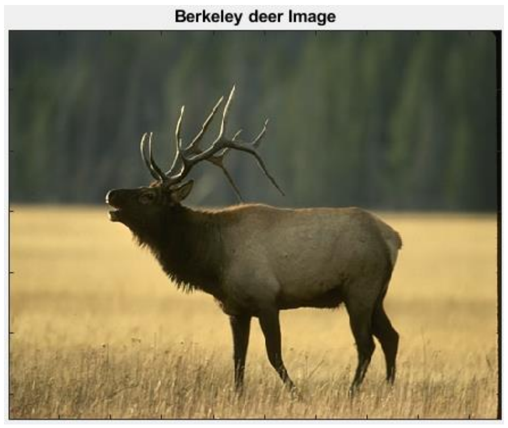

Fig 10. Berkeley_deer Image

In this algorithm, we plant region seeds, two on the background and one on the deer (desired). Normally, it is desired to obtain 3 different segments. The desired (original) segmented image is given below:

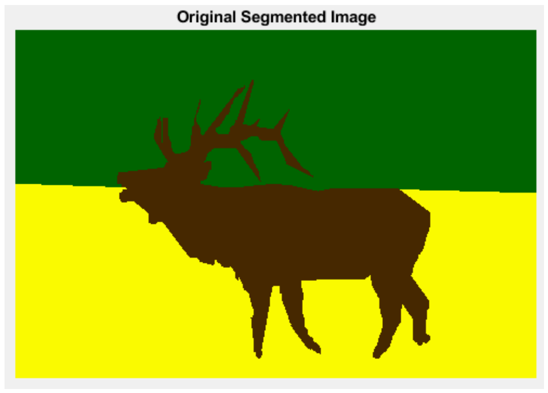

Fig 11. Segments of Berkeley_deer Image (Ground Truth)

One for the deer, one for the green background and one for the yellow background. However, one can see that the deer has two distinct body parts in terms of their colors. The head and lower body parts are darker whereas the upper body parts are lighter. Due to this, I plant two different seeds on the deer one for each part. Therefore, in total, we have 4 seeds. At the end of the segmentation process, to get a whole deer segment, the labels for the darker part and lighter part are equalized.

At the beginning, the threshold is set to 10. At each iteration, the number of labeled pixels is checked. If no change occurs the threshold is increased by 10 for the future iterations. Via this method, the algorithm continues after no pixels are left unlabeled.

As a result of the seeded region growing algorithm, we obtain the first version of our segmented image (with 4 labels) as given below:

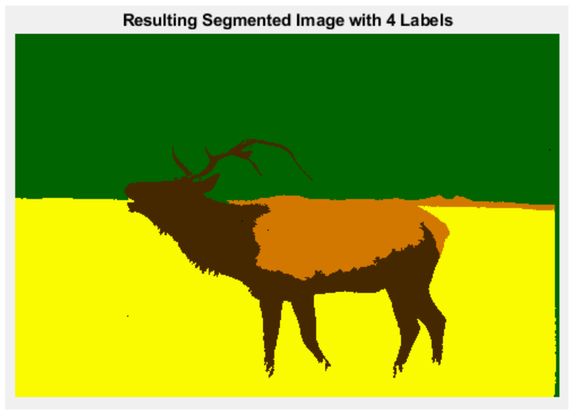

Fig 12. Resulting Segmented Image with 4 Labels (one additional)

After this, we equalize the labels of the dark part and bright part of the deer and get:

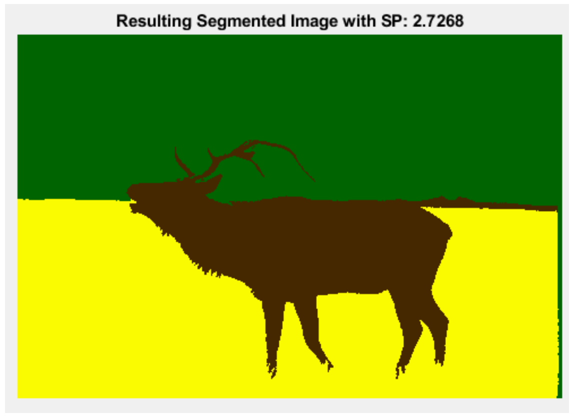

Fig 13. Final Segmented Image

To compare the resulting segmentation map with the ground truth map, the criterion $SP = \sum_{i=1}^{3} g_i = \sum_{i=1}^{3} \frac{|G_i \cap S_i|}{|G_i \cup S_i|}$, where G_i is the ground-truth region set and S_i is the segmented region set for i^{th} label. According to this performance metric, the maximum achievable score is 3. Our algorithm results in 2.7268 which is pretty good.

IV. SEGMENTATION BY CLUSTERING

In this question, the aim is to accomplish a segmentation task on Tiger image using k-Means Clustering algorithm. In total, we expect to get 4 segments.

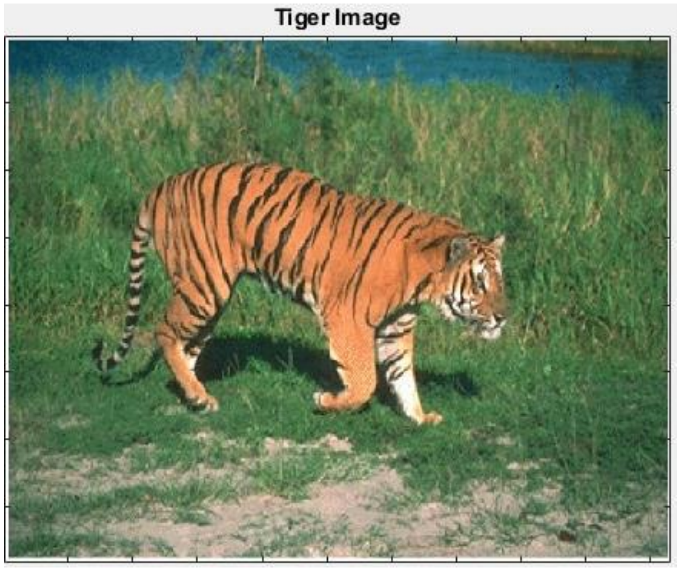

Fig 14. Original Tiger Image

A. *Segmentation by Clustering based on the feature vector f_1*

In this part of the question, k-Means Clustering algorithm is performed by my own kMeansClustering function using the feature vector $\boldsymbol{f_1} = [R\ G\ B\ ]$. The resulting segmented image is given below:

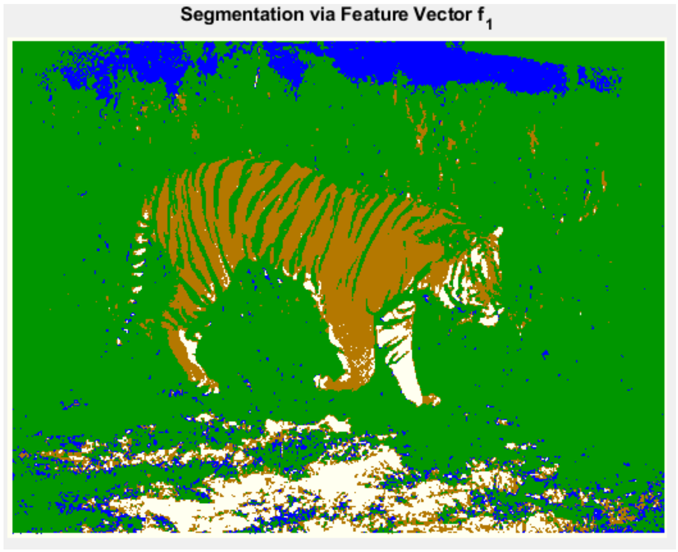

Fig 15. Resulting Segmented Image with feature vector f_1

B. *Segmentation by Clustering based on the feature vector f_2*

In this part of the question, k-Means Clustering algorithm is performed by my own kMeansClustering function using the feature vector $\boldsymbol{f_2} = [R(x)\ G(x)\ B(x)\ x\ y\ \sigma_{R(x)}\ \sigma_{G(x)}\ \sigma_{B(x)}]$. where (x,y) are the relative coordinates vis-à-vis the image center, and $[\ \sigma_R\ \sigma_G\ \sigma_B]$ are the respective variances computed from a 3x3 neighborhood, e.g., $\sigma_{R(x)} = \frac{1}{9}\sum_{-1}^{1}\sum_{-1}^{1}(f(x,y) - \mu_R)^2$.The resulting segmented image is given below:

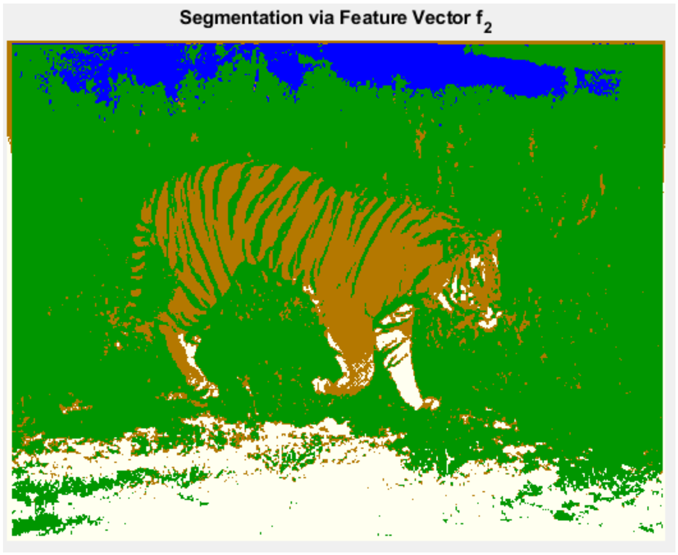

Fig 16. Resulting Segmented Image with feature vector f_2

To compare the performance of the feature vectors f_1 and f_2 we can look at the figures 15 and 16 (since no ground truth labels are provided). In figure 15, we can observe that there are small but wrong blue and orange regions. This is probably since the feature vector f_1 contains information only about the R, G, B values of a pixel not about its position in the image. In figure 16, we can observe that those mistakes are reduced since we provide more information to the feature vector f_2.

C. *Segmentation by Clustering after Obtaining Superpixels*

In this part of the question, SLIC (Superpixel algorithm) is applied using MATLAB's 'superpixels' function to reduce the pixels image down to 3000 superpixels. The boundaries of the 3000 superpixels and the resulting image after SLIC algorithm are given below:

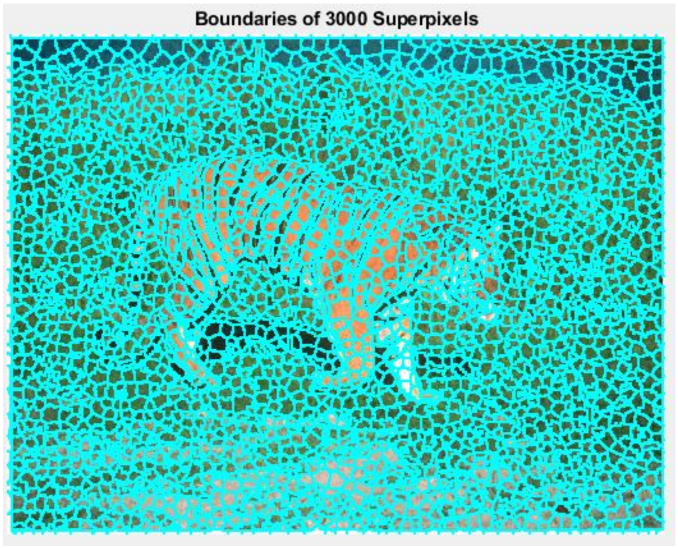

Fig 17. Resulting Boundaries of the 3000 Superpixels

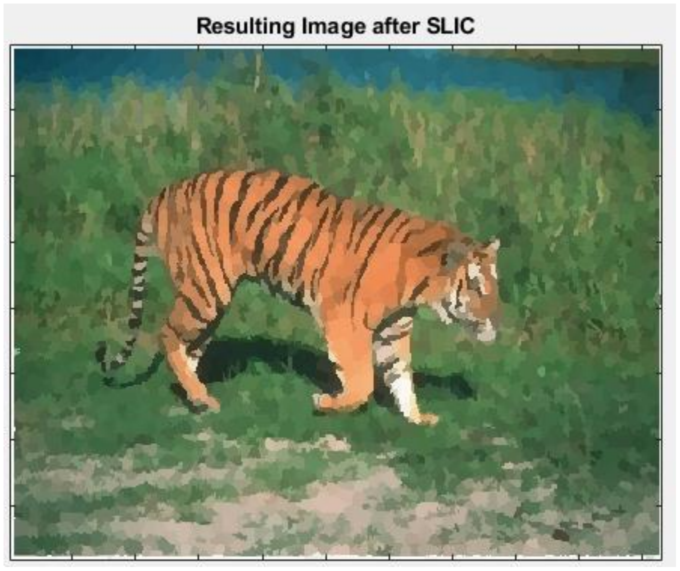

Fig 18. Resulting Image after SLIC Algorithm

Then, k-Means Clustering algorithm is performed on the resulting image by my own kMeansClustering function using the feature vector $\boldsymbol{f_3} = [R(x)\ G(x)\ B(x)\ x\ y]$. The resulting segmented image is given below:

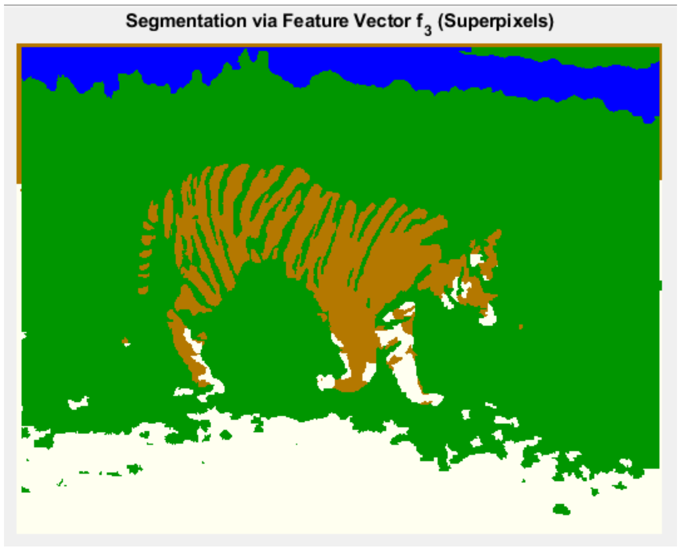

Fig 19. Resulting Segmented Image with SLIC and feature vector f_3

To compare the performance of the SLIC & feature vector f_3 approach, we can look at the figures 19 and 15 & 16. From figure 19, we can observe that the generated segments are more compact, united (less separated) and continuous than others. This is because we reduced down the pixels in the image to 3000 superpixels via SLIC algorithm.