Automatic Grayscale & Color Image Segmentation through Histogram-based Methods

Sam Johnson

***Key Words*—Time series, data mining, stock market prediction, machine learning, forecasting**

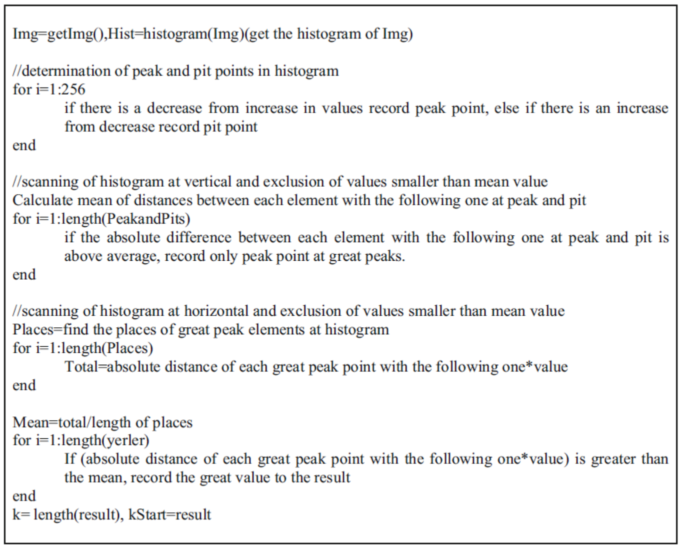
I. Introduction

For my project I decided to implement the paper *Histogram-based automatic segmentation of images* [2] and see if I could make any further contributions.

I began by implementing the author's original approach, which was on grayscale images. I then modified the author's approach slightly to apply it to color images. The results are quite interesting. I will summarize the results in the sections that follow.

II. Background

The author proposes an algorithm that should be helpful in image segmentation through the K-means clustering algorithm. The author notes that the key challenge with K-means is knowing how many clusters to have and the initial centroid locations. The proposed algorithm is given in figure 1. I did my best to implement it and will go over how in the following section.



```
Img=getImg(),Hist=histogram(Img)(get the histogram of Img)

//determination of peak and pit points in histogram
for i=1:256
        if there is a decrease from increase in values record peak point, else if there is an increase
        from decrease record pit point
end

//scanning of histogram at vertical and exclusion of values smaller than mean value
Calculate mean of distances between each element with the following one at peak and pit
for i=1:length(PeakandPits)
        if the absolute difference between each element with the following one at peak and pit is
        above average, record only peak point at great peaks.
end

//scanning of histogram at horizontal and exclusion of values smaller than mean value
Places=find the places of great peak elements at histogram
for i=1:length(Places)
        Total=absolute distance of each great peak point with the following one*value
end

Mean=total/length of places
for i=1:length(yerler)
        If (absolute distance of each great peak point with the following one*value) is greater than
        the mean, record the great value to the result
end
k= length(result), kStart=result
```

Fig. 1: Algorithm Pseudo-Code

III. Implementation

A. Image Histogram

I began my implementation by calculating the histogram of a grayscale image. This was fairly simple. I just iterated over the image and counted the number of pixels at each intensity level.

B. Peaks & Pits

Then I calculated the peaks and pits of the image. A peak occurs when the histogram increases and then decreases in value. A pit occurs when a histogram decreases and then increases in value. In other words, if the current histogram index is i, a peak occurs in this scenario:

$$hist[i-1] < hist[i] > hist[i+1]$$

A pit occurs in this scenario:

$$hist[i-1] > hist[i] < hist[i+1]$$

Additionally, I added the requirement that the order pit then peak then pit or peak then pit then peak must be maintained. In other words, if the most recent event was a peak, the next peak is ignored until a pit occurs and vice versa.
At the end of the search there was sometimes one more peak than there was pit or one more pit than there was peak. To resolve this issue, I simply took the most recent peak or pit value and duplicated it at the end of the array.

C. Vertical Filtering

Vertical filtering was the next step in the algorithm. I took the absolute difference between each corresponding peak and pit. Then, I took the average of the result. I remove all peak -pit pairs that were below this average.

D. Horizontal Filtering

I now had just a few peak values left after vertical filtering. Next I ran horizontal filtering. I calculated the horizontal distance between the peak values (the intensity values). My implementation was bidirectional. For a given peak, I found the distance from its left point and the distance from its right point and averaged them. In the case where a point is the farthest on the left, you just take the distance to the right. If the point is the farthest to the right, you just take the distance to the left.

At this point, we have our final centroids. In the rare case that there is only one centroid remaining after following the algorithm's steps, we add another one as far away from the existing centroid as possible. We can now move on to running K-means using these chosen centroids.

E. K-Means Algorithm

I used the scikit-learn package in Python [3] to run the K-means algorithm with the *max_iterations* parameter set to 1000. The package returned an array the same size as the original input image going from 0 to *n* classes. I then rescaled the array so that the values went from 0 to 1. Finally, I displayed the images in grayscale form.

F. Comparing Grayscale Image Results

I ran the algorithm on the same grayscale images that were done in the original paper. I show the results for the Cameraman image below in figure 2. We can see how the algorithm is implemented and an image is shown at each of the following steps: image histogram, all peaks, vertical peaks, horizontal peaks, and possibly adding one additional peak. At the end of the algorithm we can see that there are two selected centroids.

In figure 3 we can see the original image and the segmented image after applying K-means using the calculated centroid values. As expected, there are just two segmented sections.

Compare the algorithm steps image from figure 2 with figure 4, which is the final peaks chosen by the paper. As you can see, the results aren't the same. However, they are fairly close.

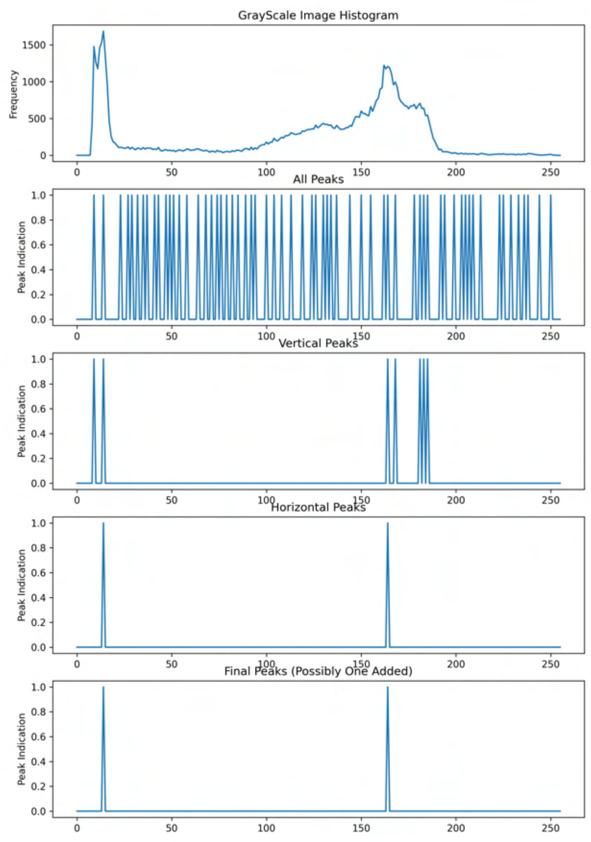


Fig. 2: Generated Images/Cameraman Peaks Image.png

Compare our segmented image in figure 3 with figure 5, which is the image shown by the paper. Other than slight differences in shading, the images appear the same. Thus, although the centroids were in slightly different locations, the results were the same.

Below, we display the algorithm step images and the resulting segmented images in no particular order.

You can compare the above images with the figure in 18, which is the author's segmentation results. We can see that in nearly all cases, our centroid finding algorithm and subsequent segmentation resulted in the exact same segmentation.

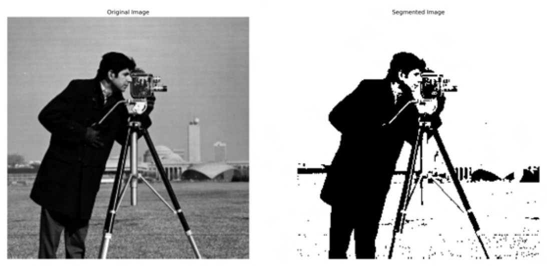


Fig. 3: Generated Images/Cameraman Segmented Image.png

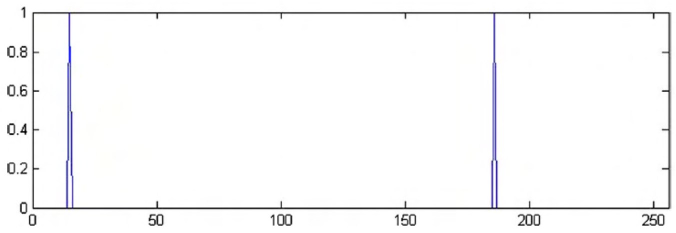


Fig. 4: Images from paper/Camera man final peaks from paper.png

IV. Color Images

The next thing we did was modify the algorithm slightly to make it applicable to color images. First, we divide the image into its channels, which are red, green, and blue. Then, we apply the same centroid finding algorithm to each channel individually. Then, we perform K-means on each channel individually. Then, we scale each channel individually so that the pixel values go from 0 to 1. Finally, we display the color images with all three channels. The results are quite interesting.

In figures 21, 20, and 19 we show the algorithm steps for the red, green, and blue channels respectively.
In figure 22 we can see the original color image and the segmented color image. I think it did a reasonably good job at segmenting it.

In the following figures we show more color image segmentation results on random images [1] while omitting the algorithm steps that reached that point. The results are once again quite interesting.

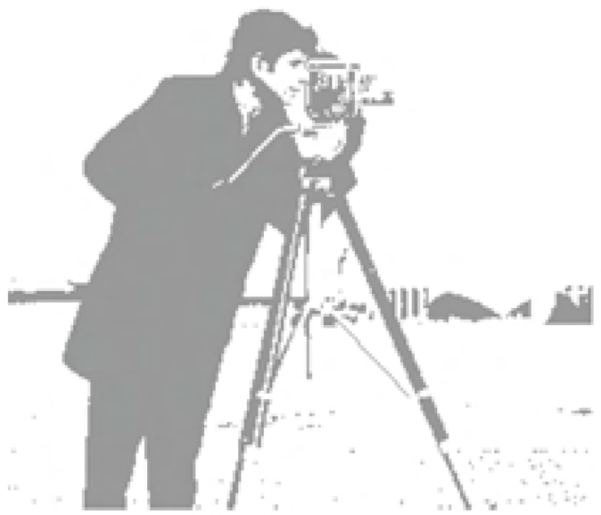


Fig. 5: Images from paper/Camera man segmented from paper.png

V. Conclusion

In this project I implemented the *Histogram-based automatic segmentation of images* paper. I described my implementation, showed the results on grayscale images, compared the results with the paper, and then created a novel implementation on color images. I showed the results on color images as well, which were very intriguing visually.

References

[1]
[2] Enver Küçükkülahlı, Pakize Erdoğmuş, and Kemal Polat. Histogram-based automatic segmentation of images. *Neural Computing and Applications*, 27(5):1445–1450, 2016.
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

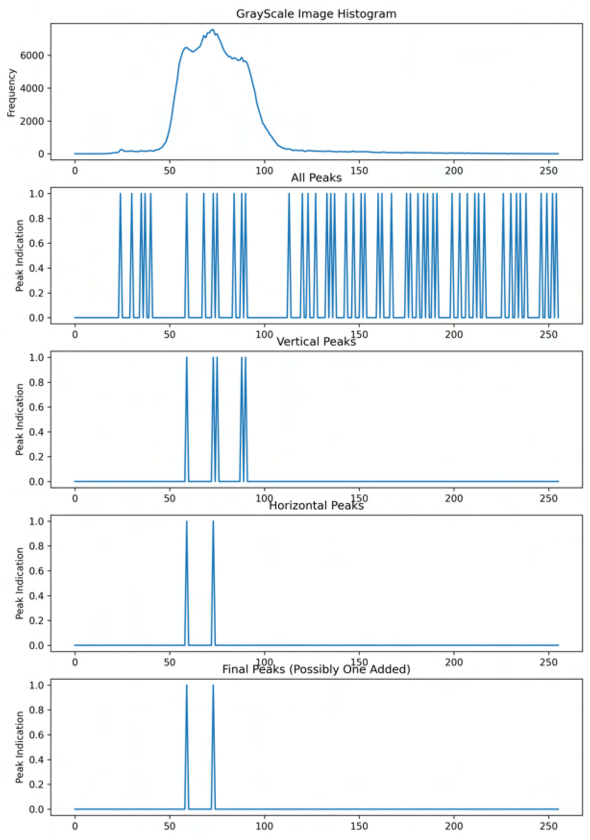


Fig. 6: Generated Images/At3$_1M4_01PeaksImage.png$

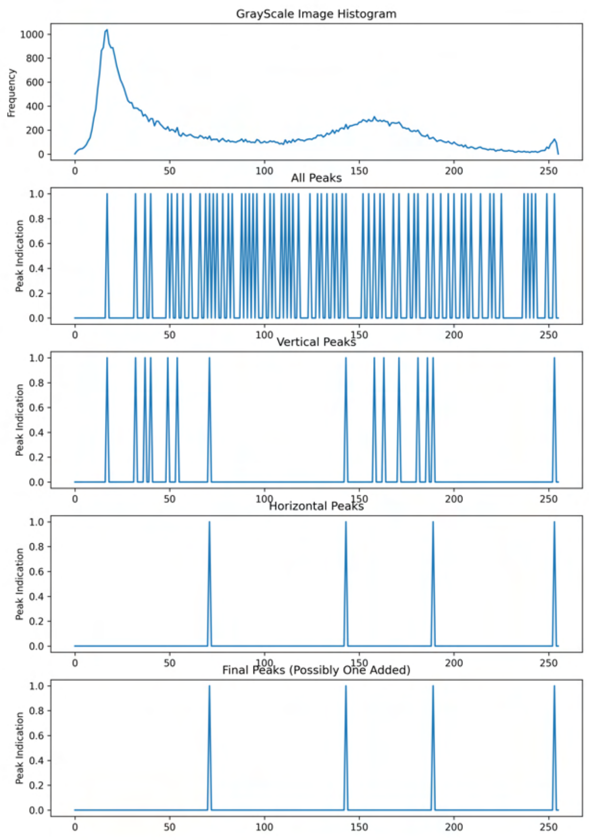


Fig. 8: Generated Images/Bag Peaks Image.png

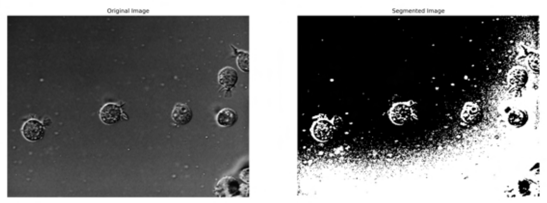


Fig. 7: Generated Images/At3$_1M4_01SegmentedImage.png$

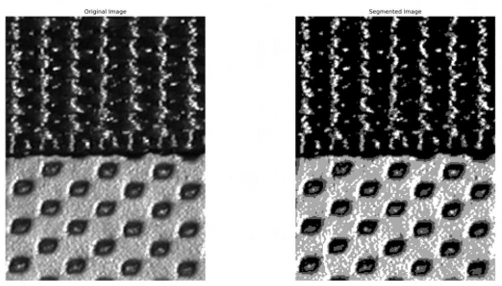


Fig. 9: Generated Images/Bag Segmented Image.png

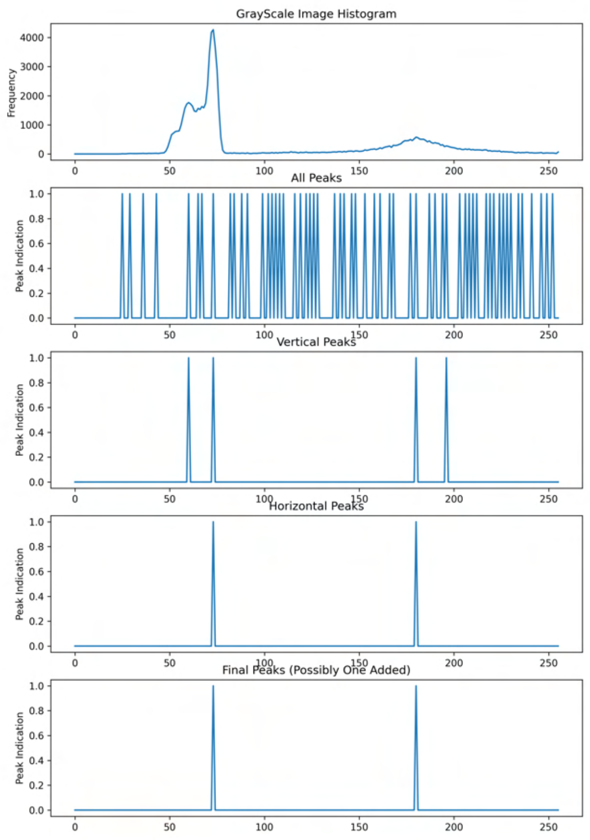


Fig. 10: Generated Images/Coins Peaks Image.png

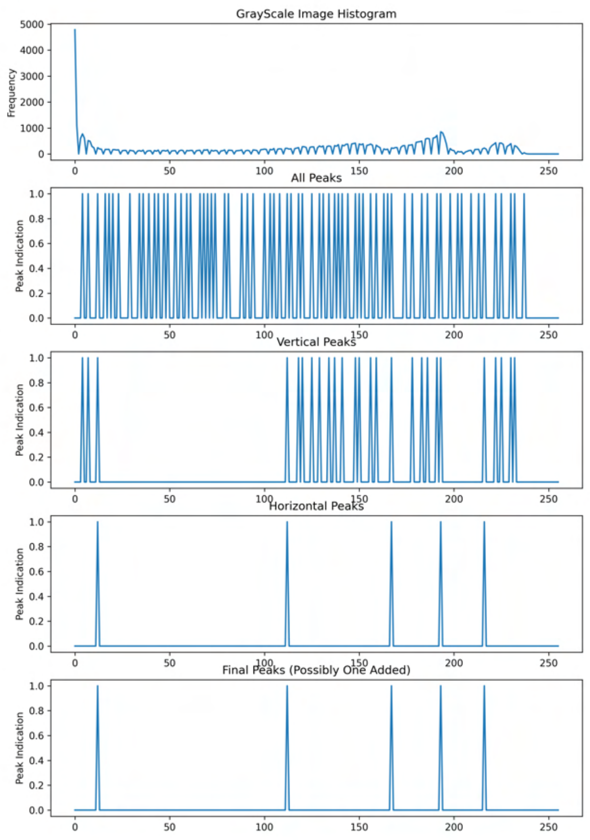


Fig. 12: Generated Images/Glass Peaks Image.png

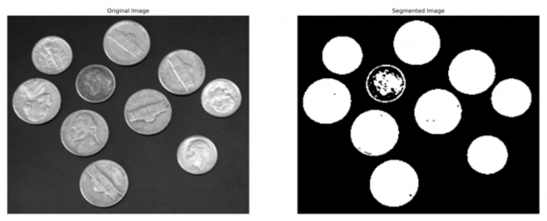


Fig. 11: Generated Images/Coins Segmented Image.png

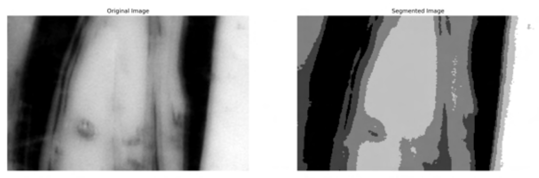


Fig. 13: Generated Images/Glass Segmented Image.png

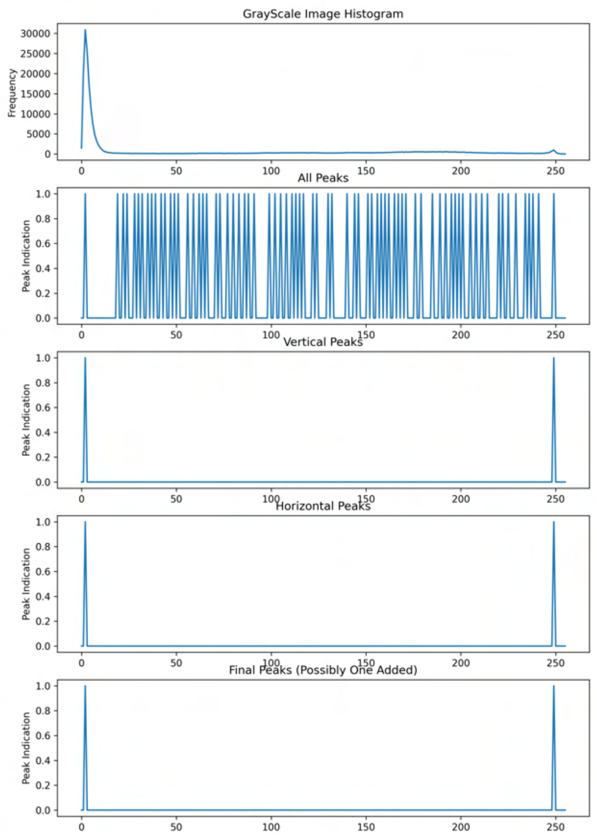


Fig. 14: Generated Images/Moon Peaks Image.png

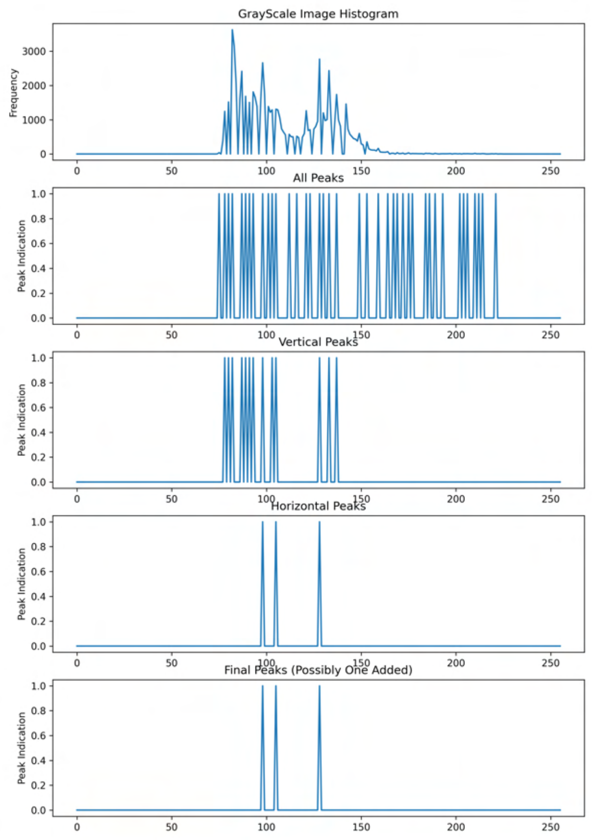


Fig. 16: Generated Images/Pout Peaks Image.png

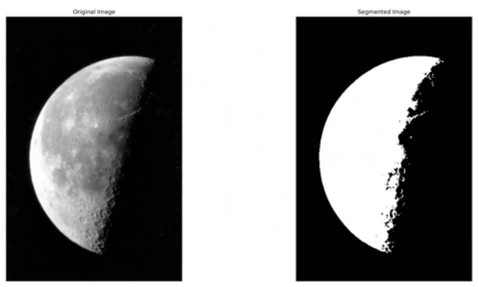


Fig. 15: Generated Images/Moon Segmented Image.png

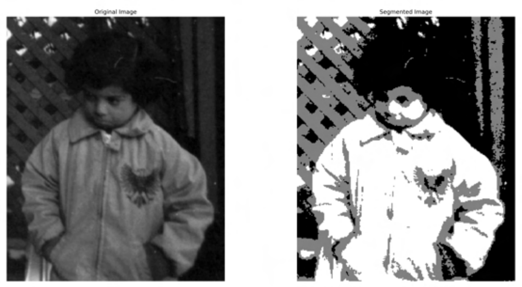
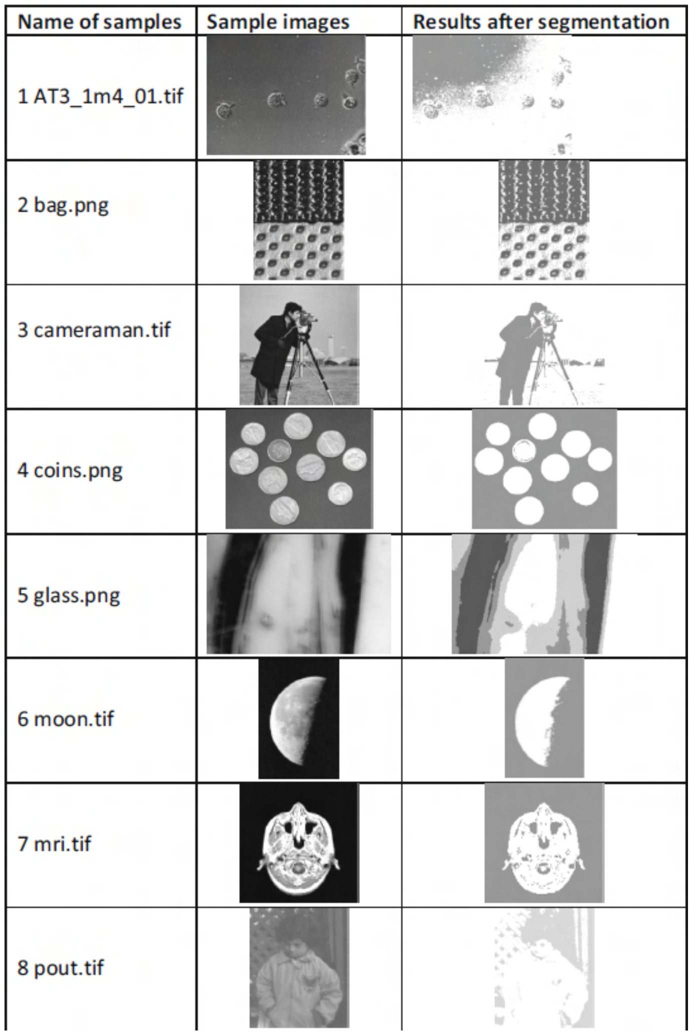


Fig. 17: Generated Images/Pout Segmented Image.png



Name of samples	Sample images	Results after segmentation
1 AT3_1m4_01.tif		
2 bag.png		
3 cameraman.tif		
4 coins.png		
5 glass.png		
6 moon.tif		
7 mri.tif		
8 pout.tif		

Fig. 18: Paper segmented images

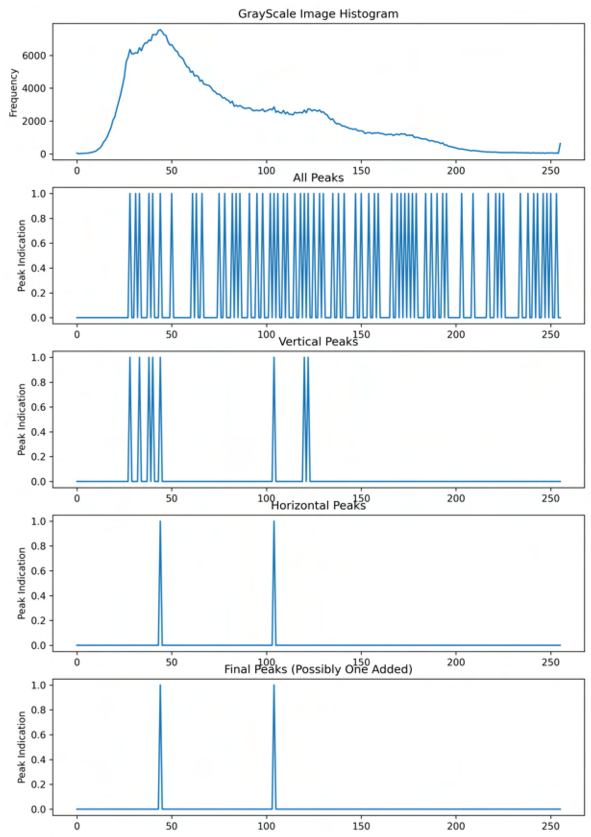


Fig. 19: Generated Images/The Galaxy Blue Peaks Image.png

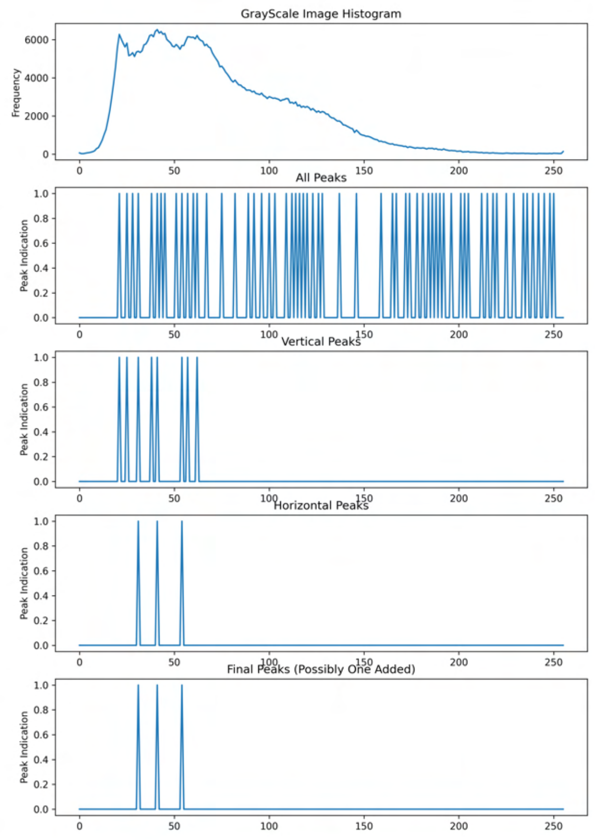


Fig. 20: Generated Images/The Galaxy Green Peaks Image.png

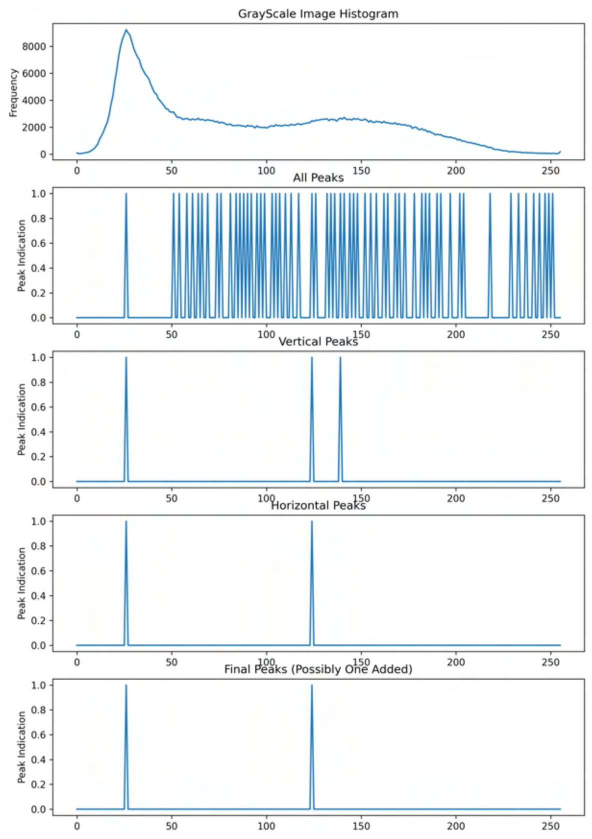


Fig. 21: Generated Images/The Galaxy Red Peaks Image.png

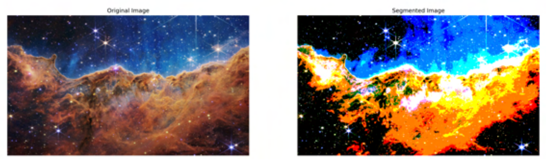


Fig. 22: Generated Images/The Galaxy Segmented Image.png

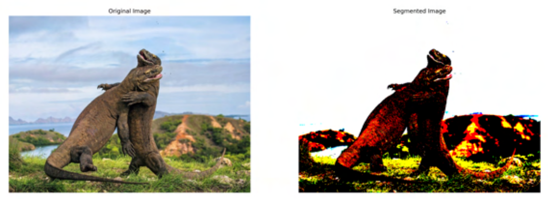


Fig. 23: Generated Images/Fighting Animals Segmented Image.png

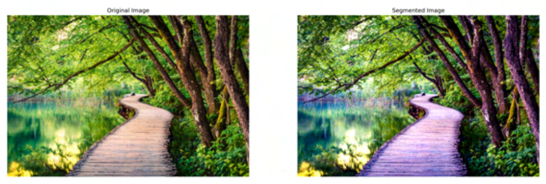


Fig. 26: Generated Images/Pond Nature Segmented Image.png

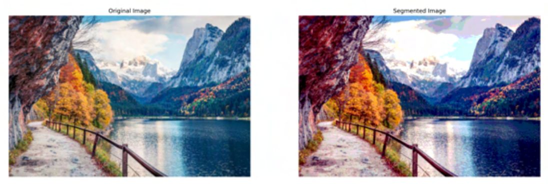


Fig. 24: Generated Images/Mountain Nature Segmented Image.png

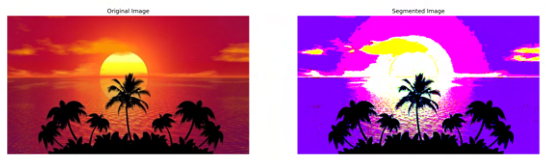


Fig. 27: Generated Images/Sun Rise Segmented Image.png

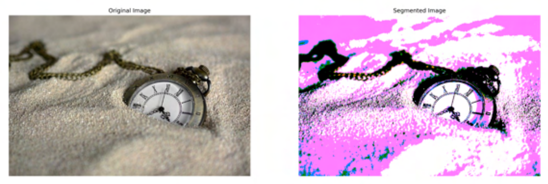


Fig. 25: Generated Images/Pocket Watch Segmented Image.png

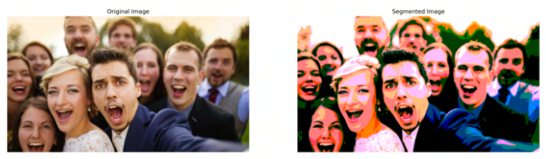


Fig. 28: Generated Images/Surprised People Segmented Image.png