MIC-Assignment-2

Introduction

For this assignment, I've chosen the following images:

1. 1.jpg
2. 2.jpg
3. 3.jpg
4. 4.jpg
5. 7.jpg

Task # 1: Computing Histograms

Task # 1 involved calculating patch wise histograms for each image. For this, I used Opencv's `calcHist()` function. A wrapper function `get_histograms` takes 2D list of image's patches, computes the histogram 1 by 1, and returns a vector of type Numpy array with `M + N` rows and `1` column.

```
def get_histograms(self):
        count = 0
        for row in range(self.N):
            for col in range(self.M):
                patch = self.image_patch_list[row][col]
                hist = cv.calcHist([patch], [0], None, [256], [0, 256])
                self.hist_list[count, :] = hist.reshape(256)
                count += 1

        return self.hist_list
```

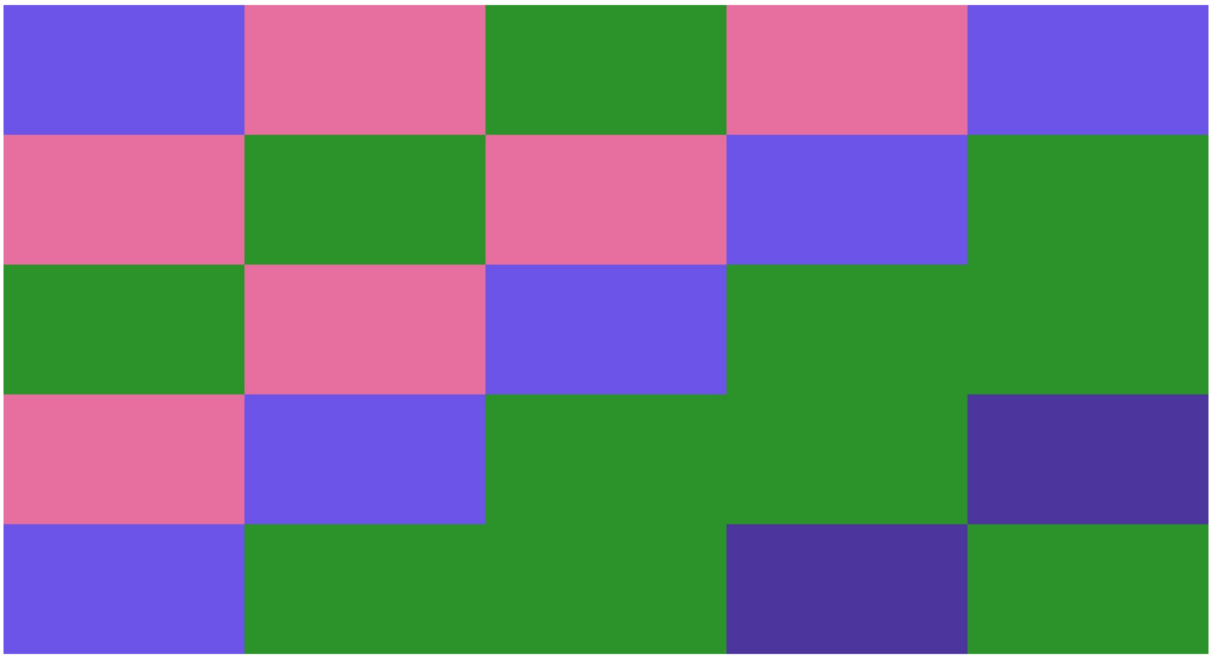
Task # 2: K Means Clustering

Using the list of images' patch wise histograms, K Means cluster is calculated. Value `K` is set as follow for all 5 images respectively.

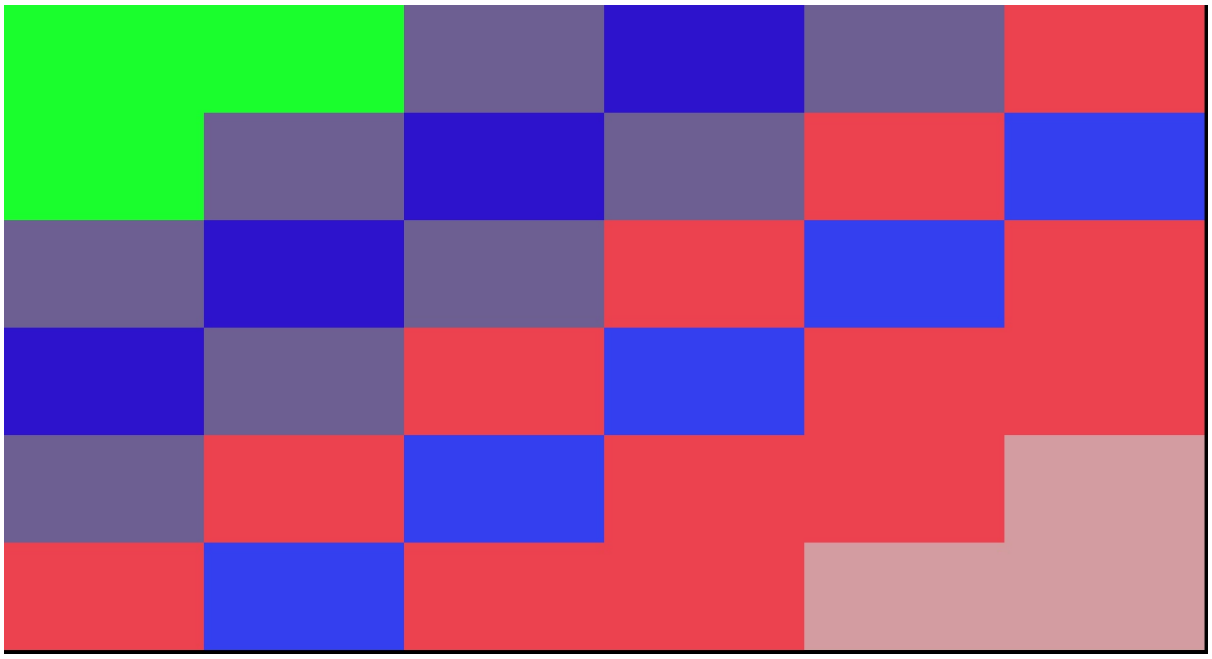
1. image 1 ⇒ 4
2. image 2 ⇒ 6
3. image 3 ⇒ 8
4. image 4 ⇒ 10
5. image 7 ⇒ 12

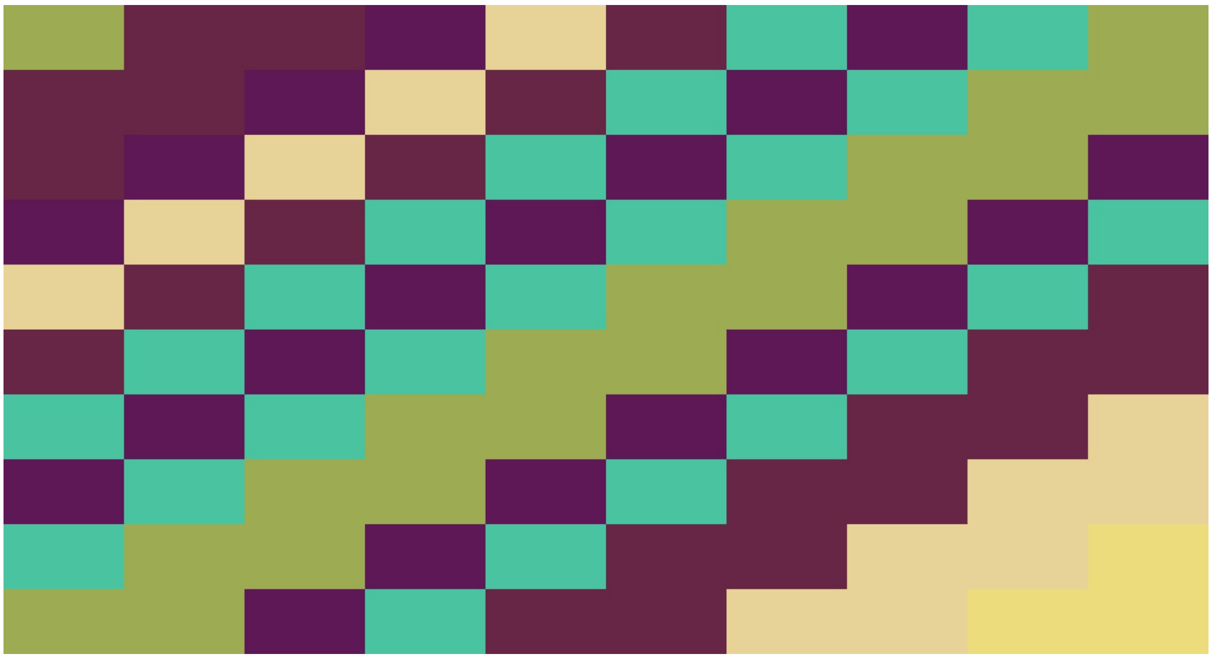
All other parameters (`init`, `random_state`) are kept same

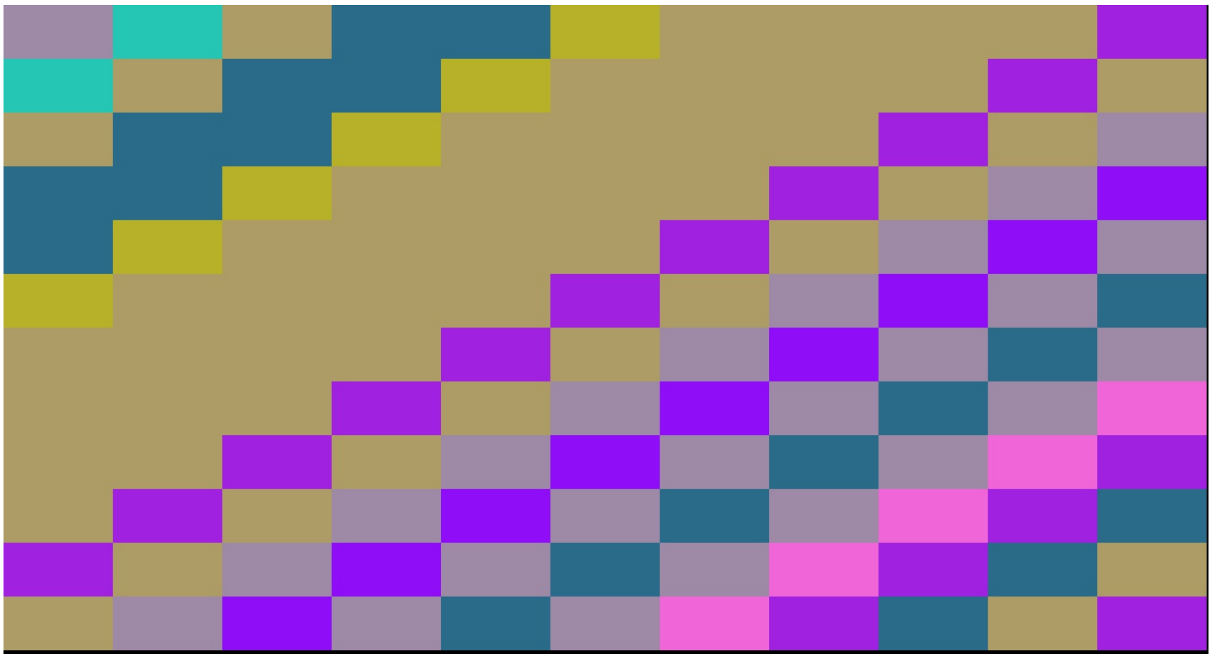
Patches are colored randomly every time the notebook is run. As a result, you will see different colors (apart from the ones shown in the saved images). Clusters results will remain the same.

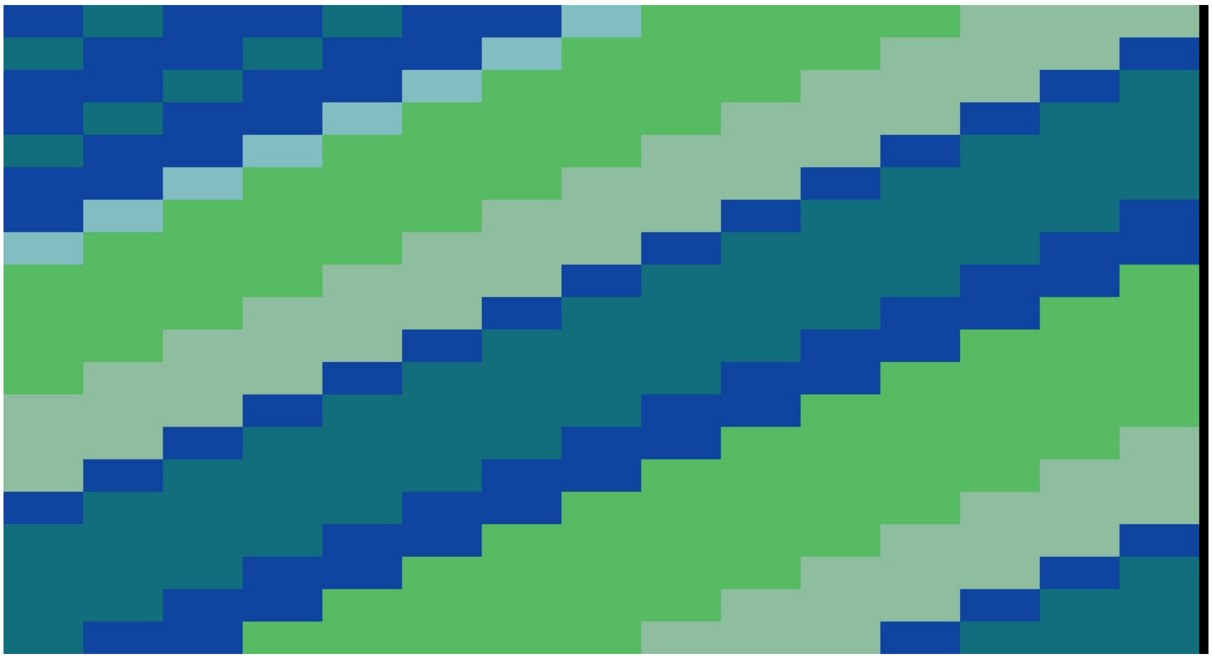


K Means result of image 1


K Means result of image 2

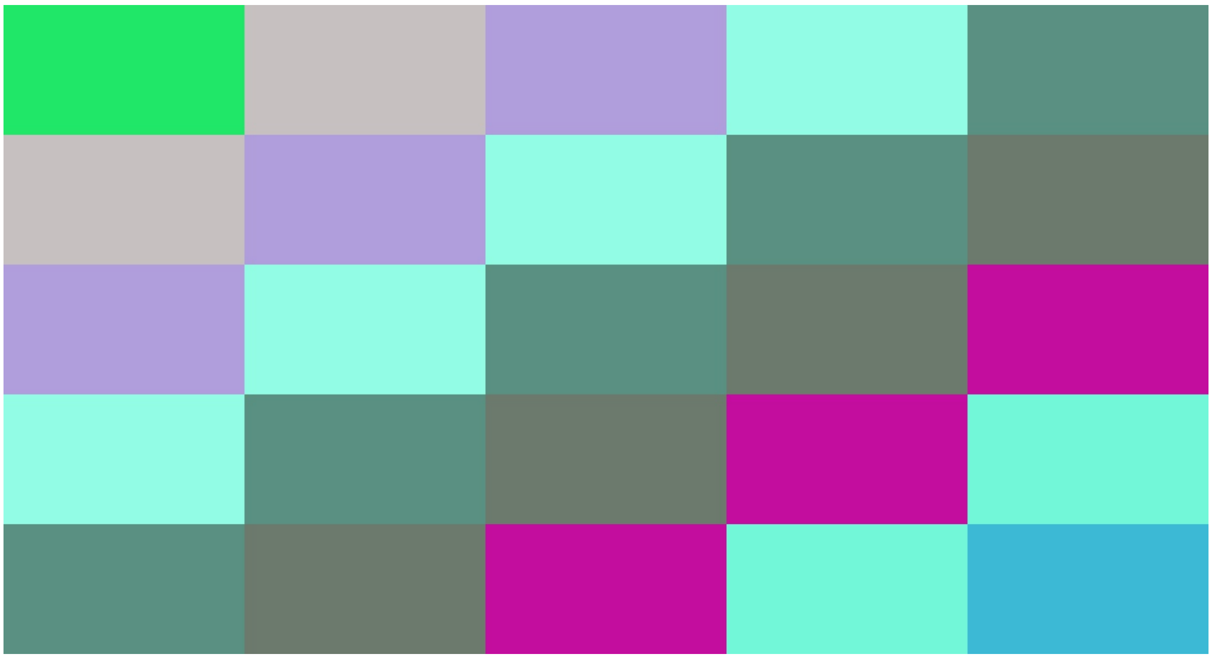

K Means result of image 3

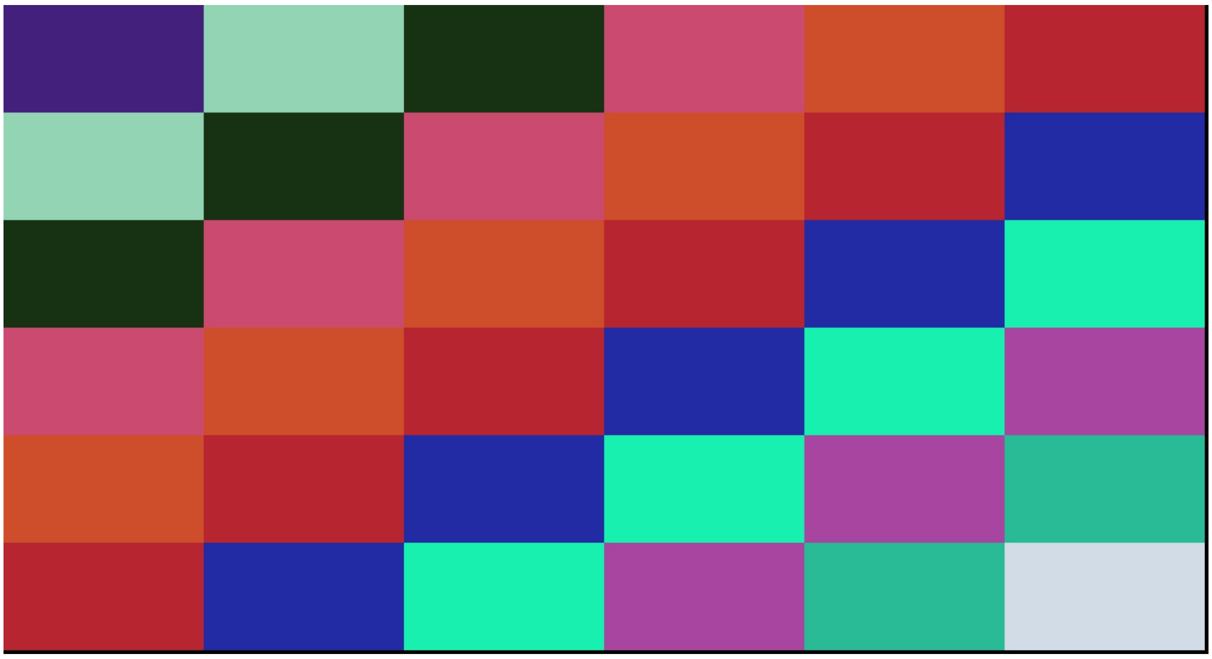

K Means result of image 4

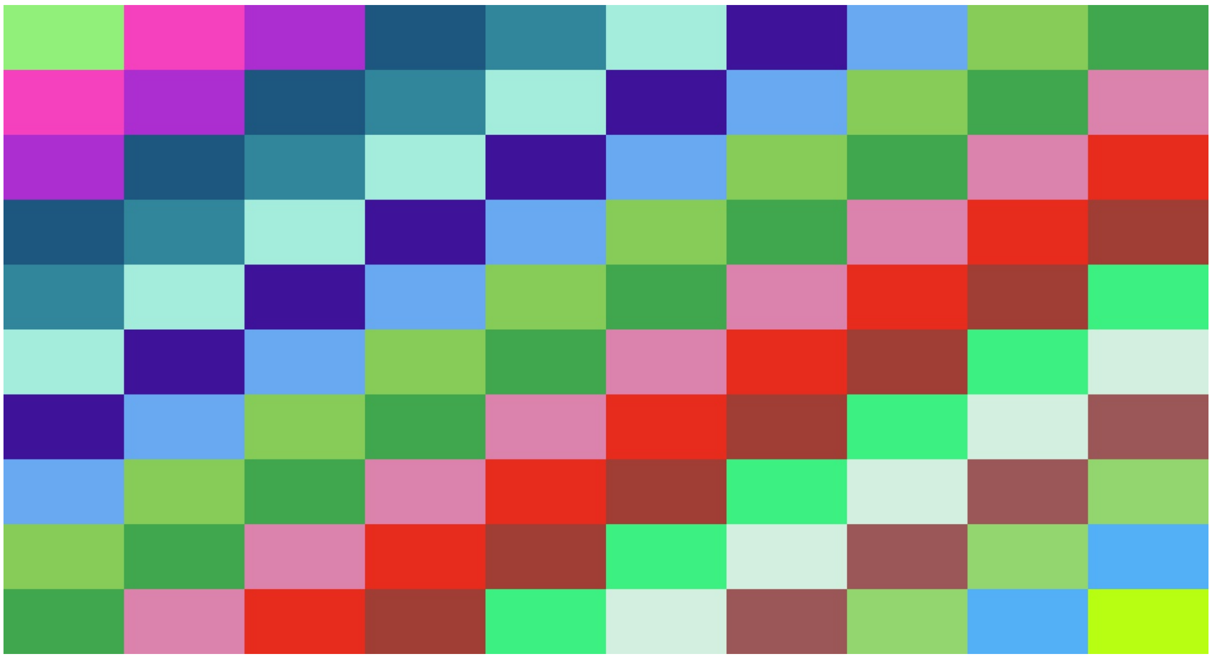

K Means result of image 7

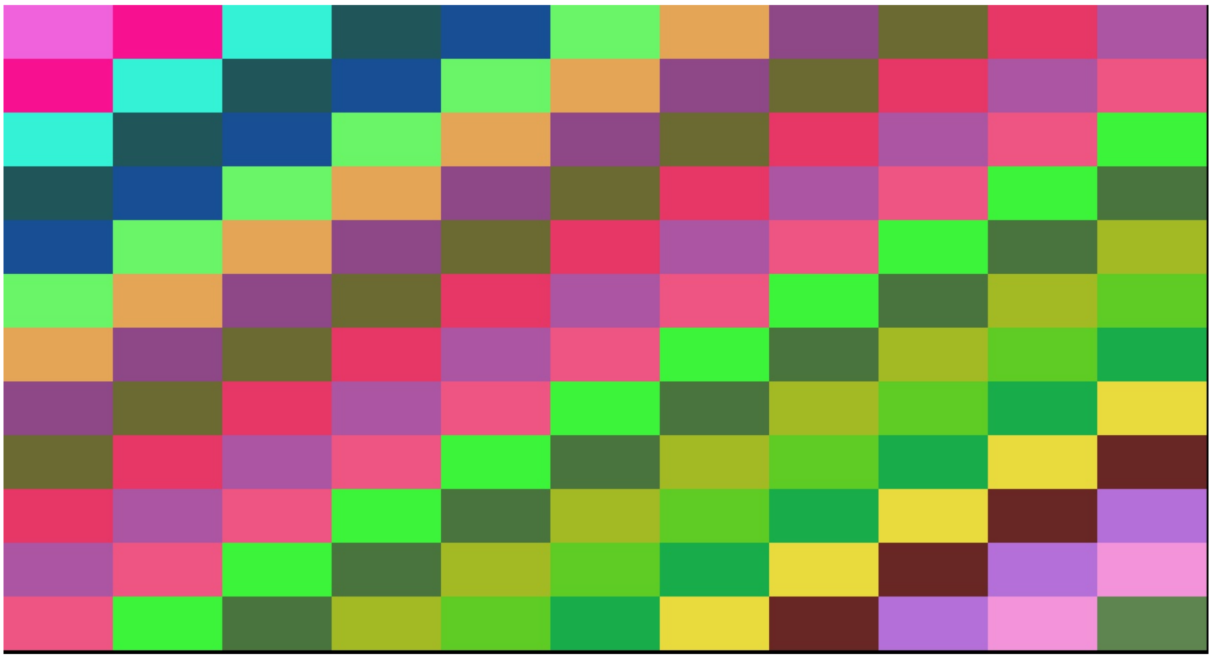
Task # 3: Mean Shift

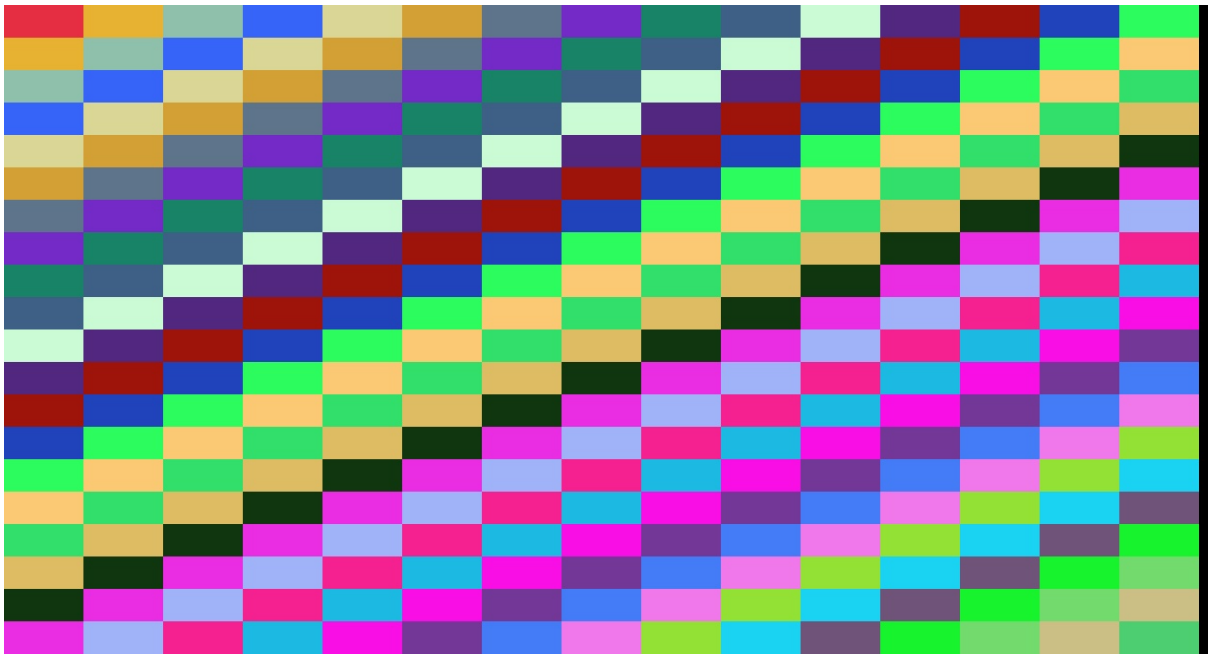
Similar to K Means algorithm, Mean Shift also takes a vector of patch level histogram, and computes the clusters. Bandwidth for images is same (`2`) for all the images. Compare to K Means, Mean Shift yields greater # of clusters (as demonstrated by the larger color pallet).


Mean Shift result of image 1


Mean Shift result of image 2


Mean shift of image 3


Mean shift of image 4



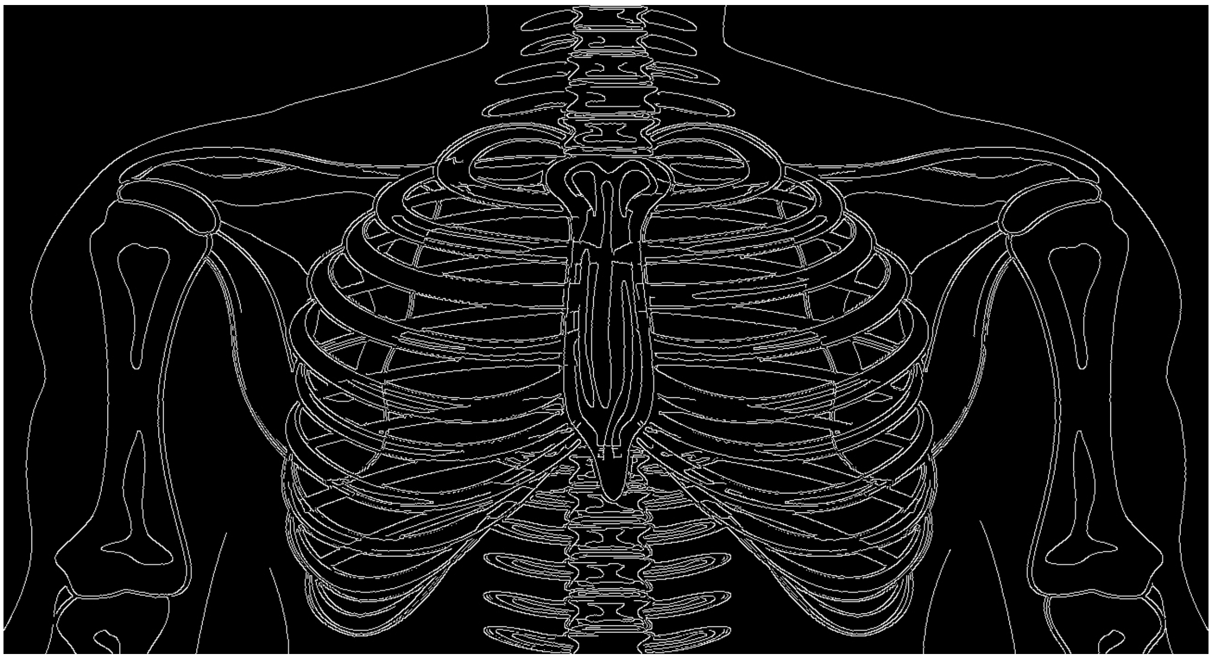
Mean shift of Image 7

Task # 4: Super Pixel Segmentation

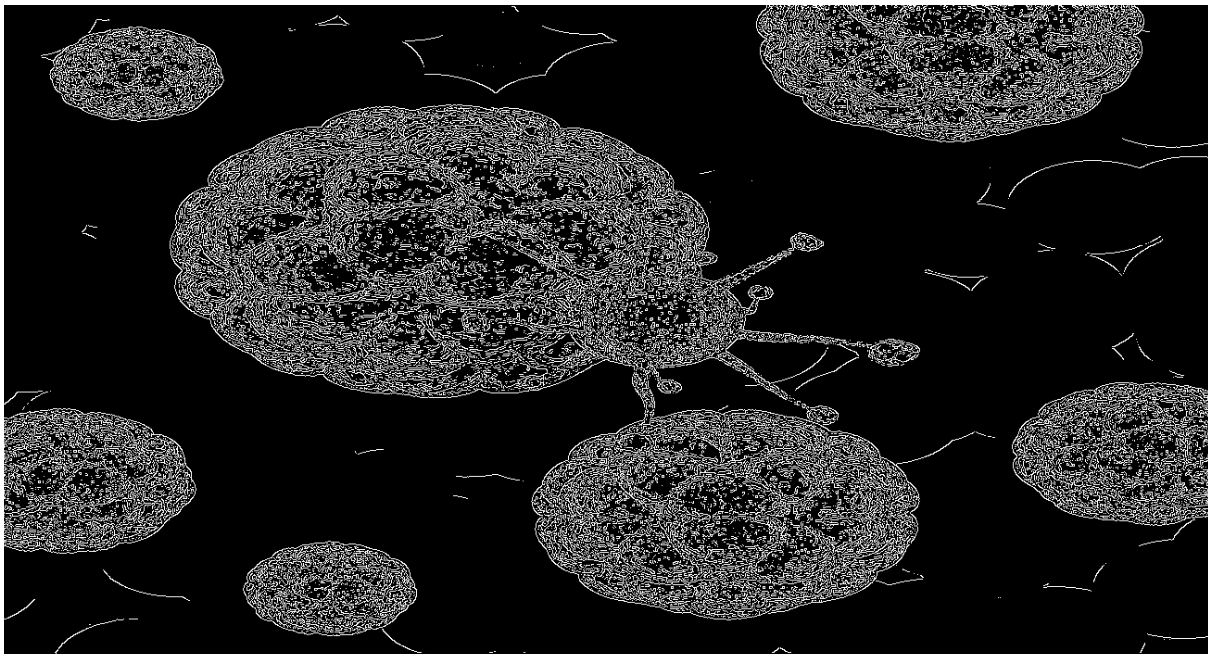
Task # 5: Active Contouring

For this task, I used Canny edge detection to find edges first. There are other methods to find edges too like **Sobel** and **Prewit** but in comparison, Canny yields relatively fine results. The only downside of Canny is its computationally expensive nature.

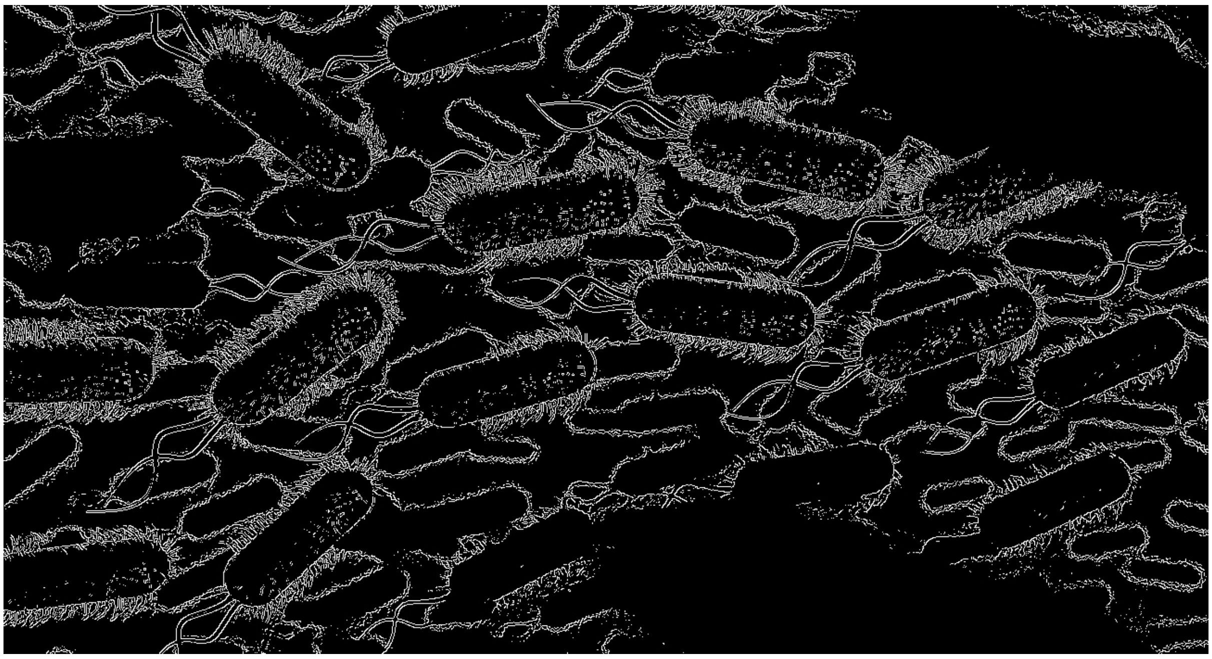
Here are the final results of Contouring with Canny edge detection. Note the edges in (below) pictures may appear distorted. For exact clarity, check out the results folder provided in the assignment.

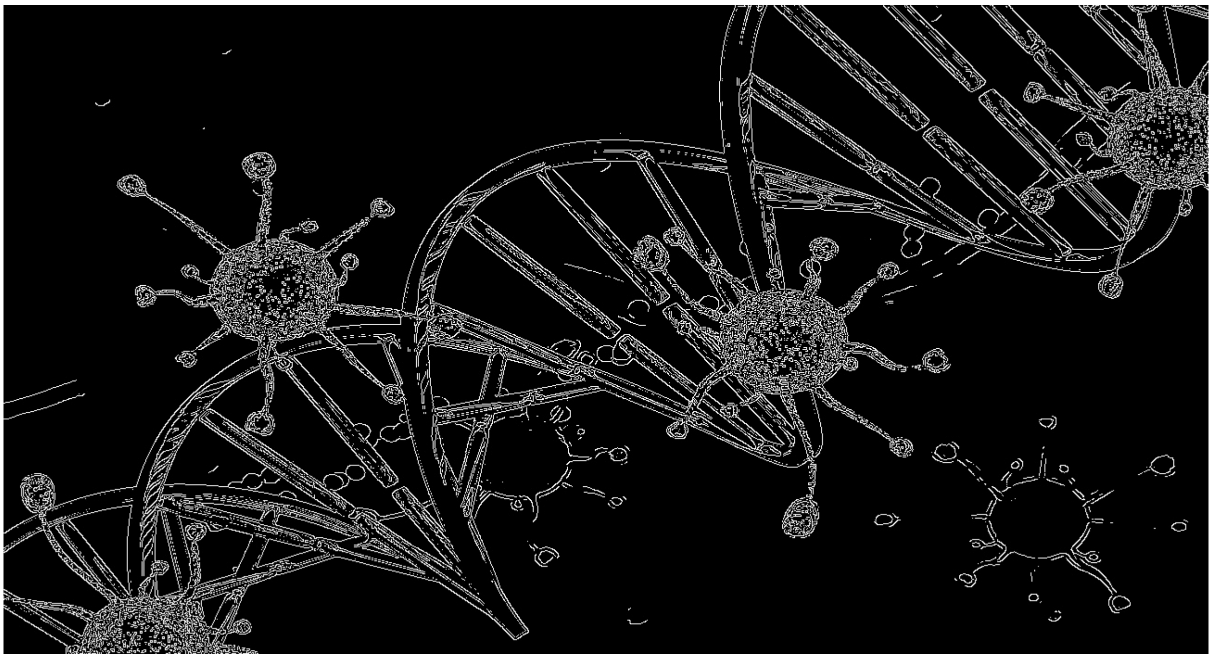


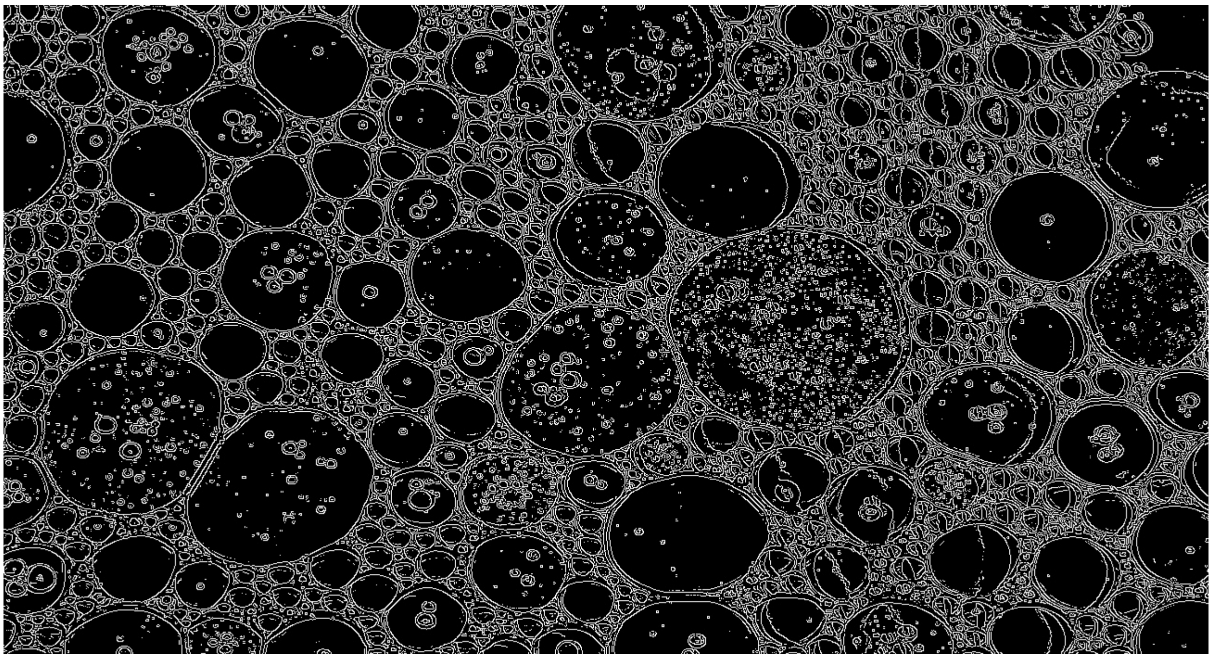
Contouring result of image 1



Contouring result of image 2


Contouring result on image 3


Contouring result on image 4


Contouring result on image 7

Task # 6: Otsu Thresholding

In order to achieve Otsu thresholding,