<u>CV Assignment 1</u>

<u>Name:</u> Meetakshi Setiya
<u>Roll No:</u> 2019253

Note: I used Matlab for the entirety of the assignment.

1.

I have added two functions Saliency_Map_Eqn3(path_to_image) and Saliency_Map_Eqn5(path_to_image, path_to_segmented_image) in the same Matlab file.

In part 1, i.e. implementing equation 3, the input to the function is a k-means clustered image (leaf_kmeans.png) with 85 colours. Thus, there are 85 centroids (colours) in the image. I have stored each of these unique centroids as a 85x3 matrix so that I can get the RGB thruples easily.
Next, I counted the frequencies and thus, the probabilities of each of these colours/centroids in the entire image. This was stored in the array color_probability.
Now, it was just a matter of looping through each of these 85 centroids and using the given formula. I calculated the distance between two colours/centroids as the euclidean distance of their RGB values. Then, I found the weighted sum of colour distance of each centroid from every other centroid with the probability of the concerned centroid as weight. I created a binary mask to replace each centroid colour in the image with its corresponding sum of distance value calculated above to get the saliency map.
Now, the saliency map had to be normalised to get the values between 0 to 1. I achieved it by subtracting the saliency matrix with its minimum element and then dividing the resulting matrix by its maximum element.
This gave me the required saliency map. The file name is leaf_saliency_eqn3.

In part 2, i.e. implementing equation 5, the input to the function is the original image (BigTree.png) and a segmented version of the original image. I have two segmented images- one with 18 segments (BigTree_segmented2.png) and the other with a 100 segments (BigTree_segmented2.png)
Here too, I first found the number of unique segments in the image. Since each segment is represented by a different colour, the number of unique colours in the segmented image gave me the answer.
Next thing to do was finding the probability of each colour in its region. I did not do a brute force implementation here. Rather, I first created a binary mask for a particular region and created a dummy matrix with the same values as the original image. I then applied the

mask to the dummy matrix and the pixels that were out of the region I assigned them -1. Then I calculated the number of unique RGB thruples for the dummy matrix (although the matrix has -1s and it isn't exactly an image) and removed all those columns. Now that I have only the interested region (temp) with non -1s and I have removed -1s from the count, the number of pixels in each region, and hence their probability was easily calculated. I stored the probabilities and the colours for each region in a cell array.
Next, I just looped through the regions and applied the given formula for each pair of regions. I again used euclidean distance on RGB to find the distance of colours from each other. Once again, I found the binary mask of the kth (concerned) region and replaced each colour in the region with its corresponding sum of distance value calculated using the equation to get the saliency map.
Now, the saliency map had to be normalised to get the values between 0 to 1. I achieved it by subtracting the saliency matrix with its minimum element and then dividing the resulting matrix by its maximum element.
This gave me the required saliency map. The file name is BigTree_saliency_eqn5.png for the 18 segment image and BigTree_saliency_eqn5_100_segments.png for the 100 segment image.

Also, I have already called the two functions in the Matlab script itself.

2.

The function name that does the deed here is Modified_Otsu(path_to_image). It takes a string argument for the image path and outputs a CSV file with 3 columns. The first column is threshold values 0 to 255. The second column is the threshold values divided by 256 to get them between 0 to 1 (since the original otsu algorithm outputs an optimal threshold between 0 to 1) and the third column has the corresponding TSS values.
Here, first I converted the image to grayscale and found all possible tuples (colours) in the grayscale image and their frequencies. Dividing the frequencies by the total number of colours gave me the probability of each colour.
Next, I looped through 0 to 255 as potential threshold values and found the total sum of probabilities for each class (below and equal to threshold) and greater than threshold. Similarly, I found the weighted mean of each class too. Now to find TSS, I added the cumulative sum of the square of difference of each grayscale pixel value in a class with the class mean. This data was stored for each threshold value (0 to 255) in an array.
Now, I found the threshold with the minimum TSS and coined it as the optimal threshold. I display both values- in 0 to 255 form and 0 to 1 form.

<u>The minimum TSS I got for the horse.jpg image was 93229886.201580 and the optimal threshold value corresponding to this TSS is 118 or 0.4609.</u>

The output is a CSV file called q2_tss_thresh.csv and a binary mask for the image stored as q2_binmask.png.

3.

Here, the function name is MinEnclosingCircle(path_to_video) that takes a string as path to the required video (shahar_walk.avi).
I first extracted all frames of the image and found the median frame using the built in median() function of Matlab. The variable video_frames_annotated will store the final video frames with a minimum enclosing circle around the man.
First, I looped through all frames in the video and subtracted each one from the median frame and found the mean of the resultant frame. This gave me a 2D array (grayscale image) which I normalised by dividing by its maximum element. I then binarized it and the resultant image just had the man in white and the background black. I then used Matlab's built in function regionprops() to get a bounding box around the man and stored it in frame_blob_bound.
To draw a circle, I needed the centre and the radius. From frame_blob_bound, I obtained the leftmost pixel coordinates, the box width, the bottom-most pixel coordinates, the box height and I used these to find the center of the man and the radius (which is maximum of either the width or the height of the bounding box).
Then , I used insertShape to draw a red circle around the man and repeated the process for each frame.
I then wrote all this to a video and the output video file is called shahar_walk_mec.avi