Computer Vision - CSE344
Assignment 2
Report

Name: Ansh Arora
Roll No: 2019022

Problem 1

For this question we were asked to implement the saliency quality measurement system depicted in the given paper's Section 3(B). It requires us to calculate two measures for saliency maps qualities – Separation Measure and Concentration Measure.

In my implementation for Separation Measure, I first of all take out the Otsu Thresholded image and find the mean and standard deviation for the foreground and background found from the Otsu measurement. Based on this, we can now find the value of L(s). To find the value of L(s), which is given to be an integration of Gaussians, I simply make use of cdf values from 0 to z* and z* to 1 which I extract from the scipy norm function. The z* values are calculated by inputting the background and foreground mean and standard deviation values in the given formula. Finally the L(s) value is inputted in the form to arrive at the correct value for Phi(s).

In my implementation for Concentration Measure, after taking out the Otsu Threshold values and thus finding the Otsu Thresholded image, to find the connected object components I made use of skimage.measure's label function. This function labels all the connected components with unique numeric IDs. Now to find the area of each connected component, I simply looped over the images and stored the area values for all the components that form a part of the foreground. This way an area dictionary was generated, which was then divided by the total area of the foreground, from which the max $C_u(S)$ value was found and was assigned as $C_{u^*}(S)$. Now the value for Psi was finally found by inputting the found values ($C_{u^*}(S)$ and the number of connected components in the foreground depicted by |O(s)|.

Finally, a dataset of 96 DL generated Saliency Maps and 96 Non-DL generated Saliency Maps was passed through these measurement functions. The DL generated Saliency Maps were garnered from the DUTS datasets ResNet passed formulated Datasets, where the Non-DL Saliency Maps were generated from OpenCV's Fine Grained Saliency Map function. Both the datasets were passed through the functions and the final values were stored in a CSV file.

B	C	D	E	F
Image_ID	DL_Separation_Measure	DL_Concentration_Measure	Non_DL_Separation_Measure	Non_DL_Concentration_Measure
0	0.999996938	0.942537455	0.943460184	0.256687916
1	0.999999999	0.979629966	0.960161909	0.338545039
2	0.999995422	0.752265006	0.952951122	0.300793768
3	1	1	0.954376093	0.295425873
4	1	1	0.960581191	0.171741971
5	0.999927742	0.834114583	0.948805608	0.175823654
6	1	0.999947154	0.949779555	0.09881471
7	0.999966857	0.958997834	0.965101906	0.157350093
8	0.999999923	1	0.949752772	0.170516967
9	1	1	0.943231604	0.4442218
10	0.999995369	0.995431799	0.968329944	0.949743677
11	0.999999974	0.999476639	0.968957145	0.548907133
12	1	0.830626299	0.952579785	0.089457602
13	0.999999999	1	0.958604004	0.136125997
14	1	1	0.964256162	0.478624519
15	1	0.999390393	0.966944797	0.225440451
16	0.999999998	0.999749248	0.941465375	0.284185779
17	0.999999939	0.99991741	0.968694619	0.192655202
18	0.999999937	1	0.958038764	0.165211279
19	1	1	0.945606724	0.184953419
20	1	1	0.950480021	0.043198341
21	0.99999251	0.991572411	0.951461394	0.03919159

Now, to see what maps overall perform better, we calculated the mean score for the different measure for both the DL and Non-DL saliency maps for the same image. The returned values were as follows –

```
Average Separation Measure for DL Saliency Maps:  0.9927298391187985
Average Separation Measure for Non DL Saliency Maps:  0.9581429170457813
Average Concentration Measure for DL Saliency Maps:  0.9621650504092302
Average Concentration Measure for Non DL Saliency Maps:  0.315190181682962
```

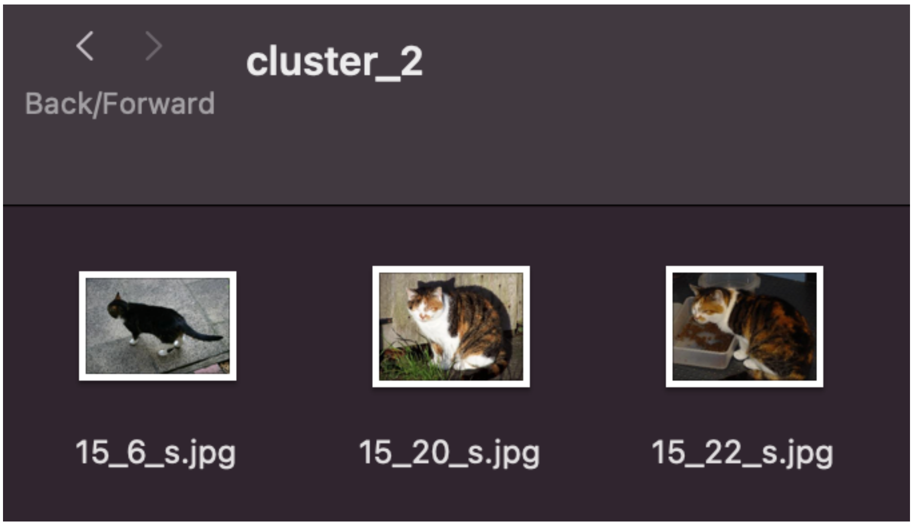
As can be observed, the average separation values for DL Saliency Maps came out to be slightly higher (almost perfect) as compared to the Non DL Saliency Maps. A high-quality saliency map should have well-separated foreground and background likelihoods like a ground-truth binary mask, which means DL Saliency Maps perform better in this case.

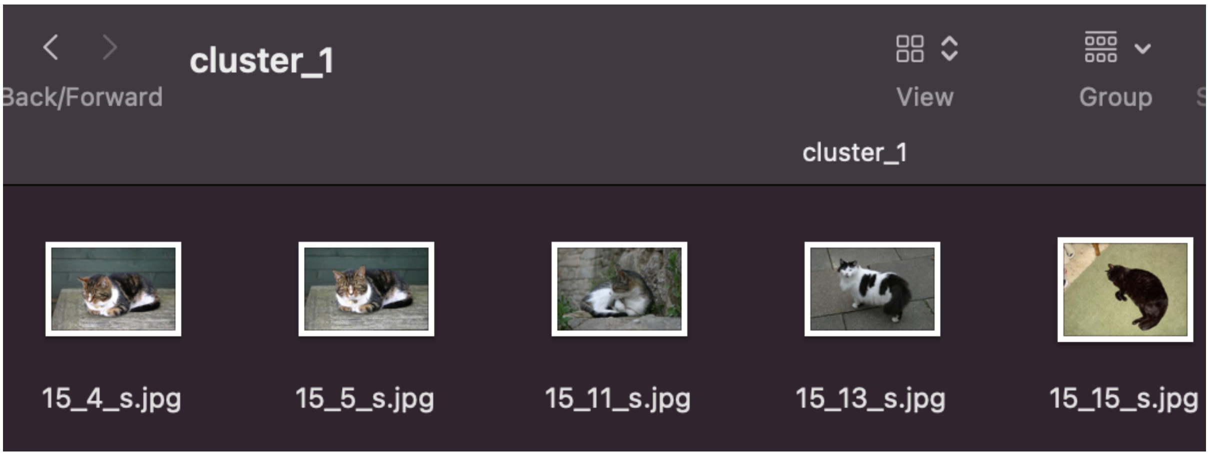
The Average Concentration values came to be much higher for DL Saliency Maps than Non DL Saliency Maps, showing that Non DL Saliency Maps perform really poorly when it comes to concentrating foreground pixels.

Problem 2

For solving the second problem, each image was divided into 3 sets of patches – one of 16 patches, one of 4 patches and one full complete image. Each patch's modified LBP Histogram was found (taking in account the change given in the assignment), from which we extract the mean and standard deviation for the LBP values. This meant that 42 features were extracted from each of the 50 images.

Now on the 50 exctracted 42-feature vectors, Fuzzy C-Means Clustering was applied and the number of k clusters provided were fitted into. After this, we simply input the images of the dataset into the predict function of the cluster module and get the cluster that each image belongs to. The images are then finally segregated into different cluster folders in the output.





Problem 3

In this problem we were asked to develop a Bag of Visual Words Model consisting of k words making use of Histogram of Gradient Features to develop the BoW feature vectors.

To do this we make use of skimage.feature's hog function. The image's were first resized to a size of 128x64, which is one that is recommended for BoVW features. After this, the image was divided into 16 patches of size 32x16, and each the HoG was calculated at 8 orientations each differing by an angle of 45 degrees. We repeated this process for all the images in the dataset and arrive at a collection of visual words. Now to formulate our visual word dictionary, we make use of K-Means. This is done using SKLearn's implementation. K number of words are found based on the input value and we find the centroid for each of the K words. Now finally, to assign the query image's features to the feature vector formed from the Bag of Words Dictionary, we just simply find the distance between the patch HoG values to all the centroids, and assign it the feature for which the centroid gives the lowest distance from. This way we are able to generate the BoW feature vector for any query image.

```
(base) Anshs-MacBook-Air:A2 ansharora$ python P_2019022_3.py
Loading images from folder:  ./data/dd
100%|██████████████████████████████████████████| 50/50 [00:00<00:00, 663.53it/s]
Calculating HOG feature vector for each of the images
100%|██████████████████████████████████████████| 50/50 [00:00<00:00, 442.70it/s]
Length of visual words:  800
Enter Value of k: 30
Number of iterations:  10
(30, 8)
(16, 8)
[14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 23, 14, 14, 14, 25, 14]
```

Problem 4

In this problem we were asked to find the K-nearest images to a given query image from an image dataset by first of all finding the image's corners using the Shi-Thomsi method and then finding the LBP patch level features for each of the corner, and then finally comparing them.

In my implementation, I first found the 10 best corners on each image using Shi-Thomsi's OpenCV implementation function, goodFeaturesToTrack. On finding these corners, a patch of 3x3 was taken for each corner, and LBP histogram is formed for each patch. The median value for each patch was found and stored, thus fetching us 10 values from each image. The process is repeated for each image.

When the query image is entered, its corners are again found and a LBP feature vector is again created from the Corner Patches. Finally, to find the k nearest images to the query image, RMSE is found between the query image and all the images in the image dataset, after which the k images with the least error are selected. These are then finally outputted in the output folder.