ASSIGNMENT 1

Sajag Agarwal (2016185)

1. Color Auto-Correlogram

Methodology:

Initially, all the images are loaded from dumped pickle files and resized to ¼ size in both directions. Now, colorspace is formed by taking a value at interval of 5 for each RGB space starting at 0. So, we have total (255/5 + 1)*(255/5 + 1)*(255/5 + 1) = 52*52*52 color values. K-Means is used to find 64 cluster centers in this colorspace. This k-means model is dumped in a pickle file. Now, for each pixel value in resized image, k-means model is used to predict the color cluster. Now, for a given distance, probability that pixel at that distance belongs to same cluster as the current pixel is calculated. Since we have 64 clusters and 4 distance values (1,3,5,7), a 4x64 vector is obtained for each image. This vector is stored and dumped for all 5063 images. Same process is repeated for query images. For a given query image, similarity between it's histogram and histogram of other image is calculated as:

– For correlogram:
  - Aggregate similarity at all distances, $k \in [d]$

$$|I - I'|_{\gamma} \equiv \frac{1}{m} \sum_{i \in [m], k \in [d]} \frac{|\gamma_{C_i}^{(k)}(I) - \gamma_{C_i}^{(k)}(I')|}{1 + \gamma_{C_i}^{(k)}(I) + \gamma_{C_i}^{(k)}(I')}$$

For a given query image, similarity with all images is calculated and the ones with highest similarity are returned.

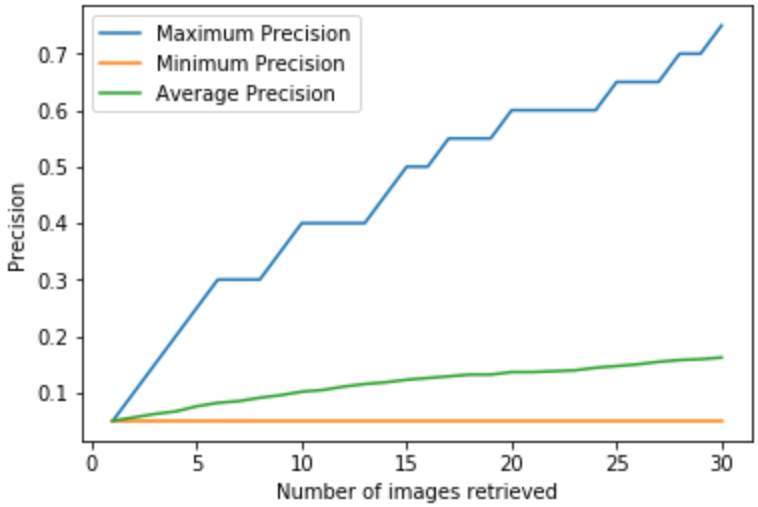
Files attached:

Q1/colorsclusters.pickle - Kmeans model to predict the color cluster of a pixel value.
Q1/correlogramnew.pickle - Dumped 64x4 color hist for 5063 images.
question1_1.py - code to retrieve images for query using color auto correlogram
question1_1a.py - code to form and dump the histograms for all 5063 images.
Q1/quercorr.pickle - Dumped 64x4 color hist for query images.

Results:

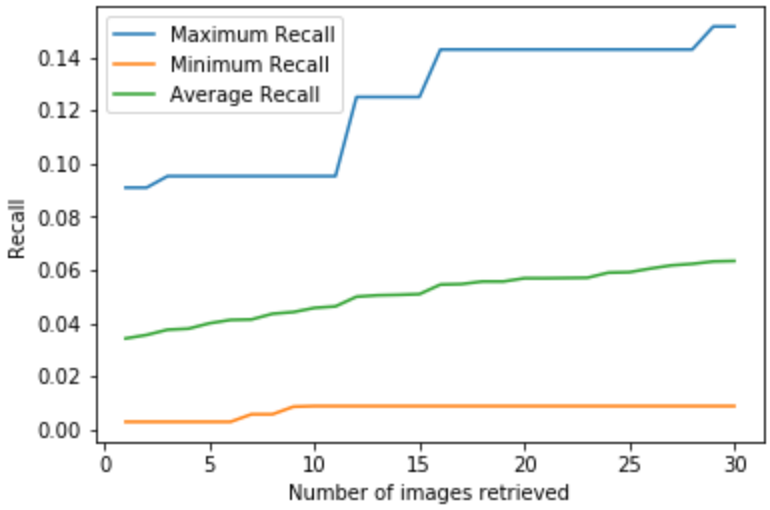
Time taken to calculate histogram per query on average: **9.71 sec**
Time taken to calculate histogram + similarity per query on average: **10.13 sec**

Precision is calculated as:
(Total number of good/junk/ok images retrieved) / (Total number of images retrieved)



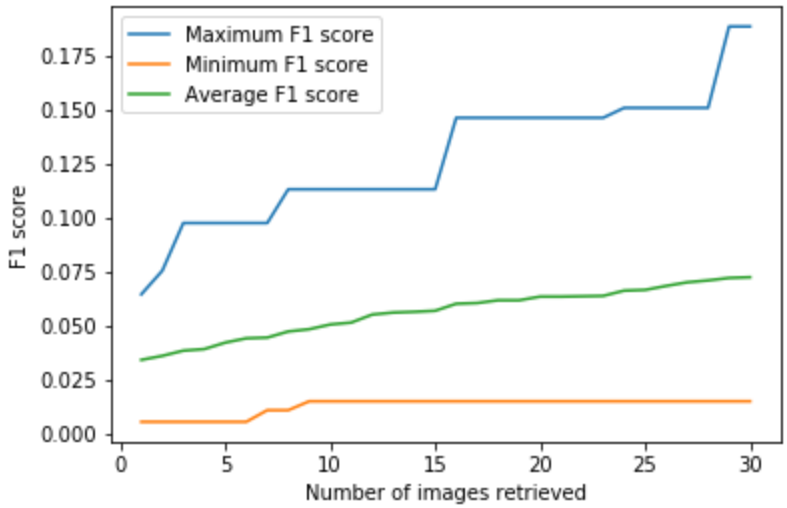
Maximum Precision is usually found for images - `'radcliffe_camera_000519.jpg' and 'oxford_002904.jpg'`
Minimum Precision is usually found for `'all_souls_000026.jpg'`

Recall is calculated as:
(Total number of good/junk/ok images retrieved) / (Total number of good/junk/ok images)



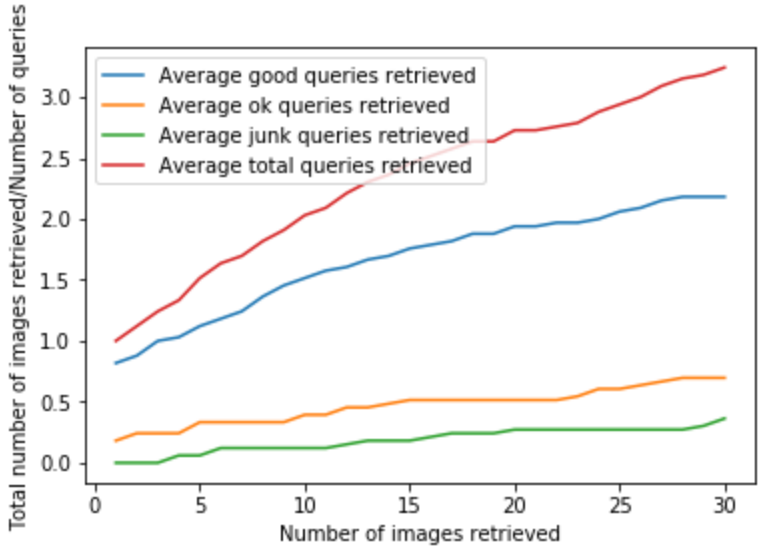
Maximum Recall is usually found for images -
`'Balliol_000051.jpg' and 'cornmarket_000047.jpg'`
Minimum Recall is usually found for `'radcliffe_camera_000523.jpg' and 'all_souls_000026.jpg'`

F1 score is calculated as:
(2*Precision*Recall) / (Precision+Recall)



Maximum F1 score is usually found for images - `'oxford_002416.jpg', 'balliol_000051.jpg' and 'bodleian_000108.jpg'`
Minimum F1 score is usually found for images -
`'radcliffe_camera_000523.jpg' and 'all_souls_000026.jpg'`

Average queries = Total queries retrieved/number of queries



Analysis:



'radcliffe_camera_000519' *'radcliffe_camera_00523'* *'oxford_002904'*

An image like 'radcliffe_camera_000519' has high precision but image like 'radcliffe_camera_00523' has low recall/F1 score despite being images of the same object clicked at different times of day. A simple analysis of images of the same object from different configuration/location shows that most of the images have either the same brightness as the left image or have different brightness from both images. Color auto correlogram fails to capture other similarities hence gives lower recall for one image but higher precision for similar image. 'oxford_002904' which is similar to 'radcliffe_camera_000519' also has high precision because of higher brightness.
Image like 'balliol_000051' has high recall because the number of images in the ground truth is very less and most ground truth images have similar color distribution like it.

2. LoG

Methodology:

Initially, all the images are read and resized to ¼ size in both directions followed by conversion to gray scale and scaling by 255. First, the image is blurred using a gaussian filter for the specified sigma and kernel size followed by laplacian filter and scaling by $sigma^2$. This is done for multiple scales for a given image. Non maximum suppression is performed on the obtained scales. For non maximum suppression, each pixel value of each scale is first compared to threshold, then compared to a 3x3 neighborhood at same scale followed by comparison to pixels

directly above and below it (that is different scales). The pixel is keypoint only if value is greater than threshold and neighbors. To fasten this process, the comparison with the neighborhood is done only if the value of pixel is greater than threshold. Now, instead of taking threshold manually, a list containing the determinant values obtained is sorted and the 6000 largest element is chosen to be threshold. This ensures that the number of blobs don't exceed a threshold as well as that there exists a minimum number of blobs for each image. Now, obtained keypoints are passed to 'blob._prune_blobs' from skimage.feature to reduce the number of overlapping blobs. Resulting keypoints are dumped to a pickle file in format [x,y,r] where (x,y) are coordinates of keypoint and r (radius of blob) is taken to be integer part of 2*sigma. To make feature descriptors, a 16x16 neighborhood is taken around each keypoint which is divided into 4x4 subregions. For each subregion, horizontal and vertical wavelet responses are taken and then for each subregion, a vector is formed as (∑dx, ∑dy, ∑abs(dx), ∑abs(dy)). Thus, for each keypoint, a 64 size vector is obtained. These vectors can be used to calculate similarity between 2 images based on euclidean distance.

Files attached:

question1_2.py - Code to extract blobs for query images
question1_2a.py - Code to extract blobs for all images
Q2/blobfeaturesall.pickle - Dumped keypoints for 5063 images.
Q2/blobimgname.pickle - Dumped names for 5063 images to read corresponding keypoints.
Q2/blobkeyfeaturesquery.pickle - Dumped descriptors for queries.
Q2/blobimgnamequery.pickle - Dumped names for queries to read corresponding keypoints/queries
blobdescriptorsquery.pickle - Dumped keypoints for queries.
Q2/queryblob -

Time taken to detect blobs per query on average: **14.3 sec**

3. SURF:

Methodology:

Initially, all the images are loaded from dumped pickle files and resized to ¼ size in both directions followed by conversion to gray scale and scaling by 255. As per the paper, sigma values (different scale values) are taken to be multiples of 1.2.

Skimage.feature's inbuilt function 'hessian_matrix_det' is used to calculate determinant of hessian at different scales for an image. Non maximum suppression is performed on the obtained scales. For non maximum suppression, each pixel value of each scale is first compared to threshold, then compared to a 3x3 neighborhood at same scale followed by comparison to pixels directly above and below it (that is different scales). The pixel is keypoint only if value is greater than threshold and neighbors. To fasten this process, the comparison with the neighborhood is done only if the value of pixel is greater than threshold. Now, instead of taking threshold manually, a list containing the determinant values obtained is sorted and the 6000 largest element is chosen to be threshold. This ensures that the number of blobs don't exceed a threshold as well as that there exists a minimum number of blobs for each image. Now, obtained keypoints are passed to 'blob._prune_blobs' from skimage.feature to reduce the number of overlapping blobs. Resulting keypoints are dumped to a pickle file in format [x,y,r] where (x,y) are coordinates of keypoint and r (radius of blob) is taken to be integer part of 2*sigma.

Files attached:

question_1_3.py - code
Q3/surfqueryblob - This contains .jpg of all the query images with surf detected blobs
Q3/surf_keypts_all.pickle - Keypoints of all the 5063 images
Q3/surf_all_name.pickle - Name of image of corresponding keypoints
Q3/surf_kpt_query.pickle - Keypoints of all the 33 queries
Q3/surf_query_name.pickle - Name of query image of corresponding keypoints

Time taken to detect blobs per query on average: **3.94 sec**
SURF takes significantly less time than LoG to find blobs as expected.