Assignment 1 - Pranjal Kaura 2017079

The images were added to the drive. All the code was run on Colab.

The basic workings of the model:

1. **Correlogram**: The images are fed to the model one by one. To increase efficiency, the colors in the image have been grouped and mapped to 512 different colors. I've used 5 different values for distances, which are [3,5,7,9,12]. Pixels in the image are then traversed. To increase efficiency, a jump of X/10 and Y/10 have been added so that our model doesn't have to look at every pixel val. 8 neighbours are checked for a given pixel,and for a given cur_distance.
   Finally I've saved an array for every image where array[dist][col] gives the prob to find the color Col at distance dist.
2. **Blob**: In the blob model, I've saved a 2D array of shape (2000, 3) wherein I've set a low threshold value and extracted the top 2000 i.e. most interesting 2000 blob points. I then save the pixel value and the coordinates of these 2000 points i.e. (image[x][y], x, y)

Matching:

1. For similarity matching, I've used 2 different metrics for defining the distance between 2 images. They are:
   a. Sum of the absolute value of difference between 2 features.
   b. Squared sum of differences of the 2 features.
2. Further the images have been ranked according to the distance, and the top 50 of them have been extracted.
3. To check the results, the top extracted images have been checked with the good ok and junk files given to us.

Precision: Relevant_Images / Total_Images_Extracted
Recall: Relevant_Images_Extractced/Total_Num_Relevant

	Average	Maximum	Minimum
F1_Score	0.044841/0.04827	0.091503/0.125	0.01801/0.02469
Precision	0.06060/0.06545	0.31/0.32	0.02/0.021
Recall	0.0664/0.068	0.181818/0.16666	0.01503/0.01436

As can be seen from the table, there is very little difference(except f1 score to a little extent) between the 2 metrics used for distances. This is very intuitive too, because the 2 metrics differ only in magnitude.(one is an abs sum, while other is a squared sum).

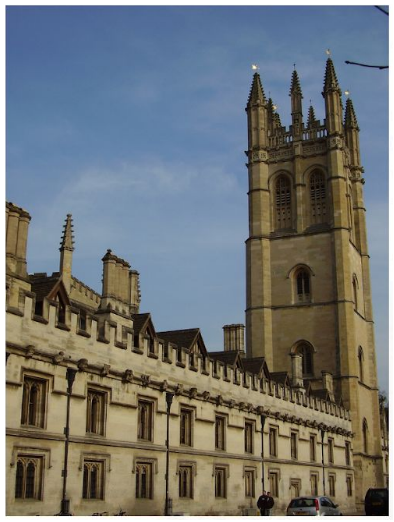
Percentages:

1. perc_good [0.4, 0.4] i.e. 40%, 40%
2. perc_ok [0.2857142857142857, 0.2857142857142857] i.e. 28%, 28%
3. perc_junk [0.061224489795918366, 0.08163265306122448] i.e. 6%, 6%

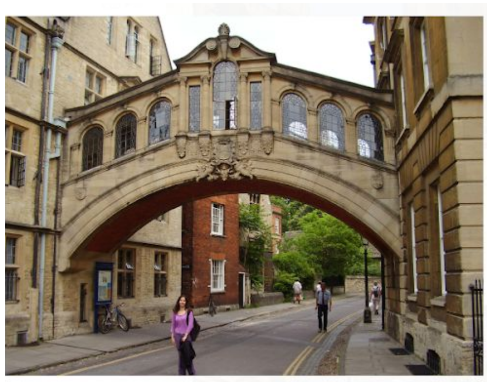
Average time:

average_time [10.627218393939401, 0.06467560606059082]

Max f1 score was recorded on image magdalen_000058. This could be because the image has very small colour variations. It's basically dominated by 2 colours(blue and light brown). Thus distortion due to grouping into 512 bins is very less. There is very little probability of existence of another image that has these colours to such extent, and which is not similar to this image.



Min_f1 score was recorded on image hertford_000015. As can be seen from the image, it has a high number of colour variations. Thus grouping into 512 bins could have caused distortions. Also the presence of white background could have further caused problems, as the model would have learned that there should have been a blue background there.



Very low recall and precision values are seen. This is a clear indication that our model is not performing well on the given dataset. Following could be the reasoning:

1. Resizing to **(512, 512)** destroys a lot of information, as input images have size in the range of 750X1000.
2. The scale of the image is distorted as the image is resized by different factors vertically and horizontally to fit into (512, 512)
3. Input images have colours in the range of 256X256X256. To make the model efficient we've thresholded/grouped the colors into ONLY **512** groups. This further leads to loss of information.
4. I'm only checking eight neighbours of a pixel, to build my model. A better model could check more pixel values.
5. The images in the dataset can not be properly differentiated on the basis of localised colour detection techniques.
6. Also possible(though unlikely) :- given good, ok and junk values are not so accurate.

On further analysing the output through manual checking, I found out that the predictions given by our model made some sense. For e.g when it encountered a query image with a road in it, it displayed outputs wherein there was a grey/light black elongated patch in the image(looking like a road). Similar findings were in the case of images of buildings. Matched images had some kind of building/structure in them having similar colour.

Link to the Blob features:
https://drive.google.com/drive/folders/1b4O0swQG30AqjwN9mx64vtYF54DPpSn1?usp=sharing

References:

1. https://projectsflix.com/opencv/laplacian-blob-detector-using-python/
2. http://slazebni.cs.illinois.edu/spring18/assignment2_py.html
3. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html