MCA Assignment 1

Anish Madan, 2016223

February 2020

1 Q1 - Color Auto Correlogram

A color auto correlogram is a method to compute image features. It was developed as an improvement over the histogram as it incorporates the spatial information into its representation to some extent.

1.1 Method

We uniformly divide the RGB color space into bins by choosing an interval length of 4. Hence, this reduces the tota possible colors from 256^3 to 64^3. We then apply K-Means to these colors to obtain 64 color clusters. These color clusters are used to quantize the RGB colors of a query image into these 64 clusters. Also, we consider the distances= {1,3,5,6} for constructing the autocorrelogram. All other details including the method of matching are same as those mentioned in slides.

1.2 Results and Inference

For computing precision and recall, we compute them at top-k values. Hence, we consider values of k=[200,300,400,500,1000] for our experiments. We tabulate results (1) obtained on query images provided and try to explain them.

- F1 Score: 0.1359
- Percentage of Good in Preds: 42.857
- Percentage of Ok in Preds: 14.2857
- Percentage of Junk in Preds: 42.857
- Time Taken per Query: 7.53s

The F1-score is computed using the average precision and average recall values for each image. We see that the good and junk percentages are similar but ok score is low. One reason could be the clustering of our colors. If we could make color clusters based on the colors present only in the training set, then maybe

Results obtained for Correlogram			
Metric	Mean Min	Mean Max	Mean Avg
Precision	0.007	0.015	0.01
Recall	0.1	0.2333	0.14667

Table 1: We see that the precision values are not very good. This is expected as the spatial information incorporated in correlogram is very limited and it is still a global representation as opposed to computing local features and them matching. Similar explanation follows for recall values. Another expected observation noted was that precision decreased for a particular image with increasing k, and vice versa for recall. This is primarily due to the formulation of precision and recall(which have number of top-k predictions i.e k, and number of ground truth images respectively).

we could boost the good, ok, junk scores(in decreasing order), but that would not be suitable for test images which have not been seen by us, as the clustering would not be suitable for it(for unseen colors).

2 Q2 - Scale Invariant Blob Detection

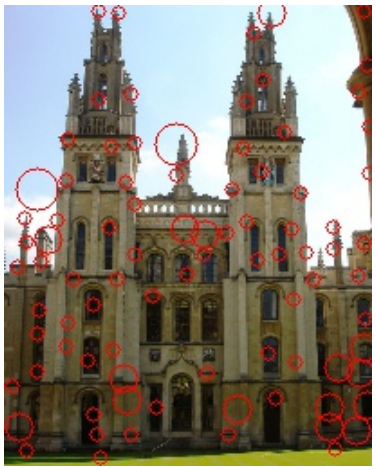
2.1 Method and Visualization

We use scale normalised LoG filters (and multiply them with σ^2 for scale invariance) for detecting blobs. We use initial value of σ as 1 and the multiplicative factor $k = 1.5$. We use 8 scales and hence increase the value of σ by k at every scale. We convolve this filter over the image, compute local maxima, do non-max suppressions and reduce overlap among generated blobs of various sizes.and then after keypoint generation, reduce the number of blobs by decreasing the overlap among blobs. We visualize some of the results obtained in figure 1

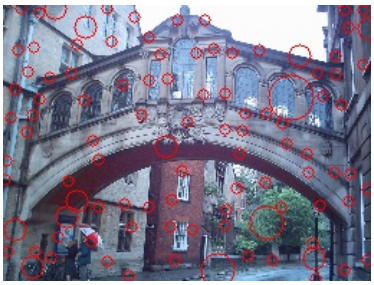
3 Q3 - SURF Keypoint Generation

3.1 Method and Visualization

SURF or Speeded Up Robust Features are an improvement over SIFT in terms of speed. It uses an integer approximation of the Hessian Matrix which uses the integral image. After obtaining keypoints, we remove overlaps to finalize the our keypoints for the query images. We visualise the results in 2

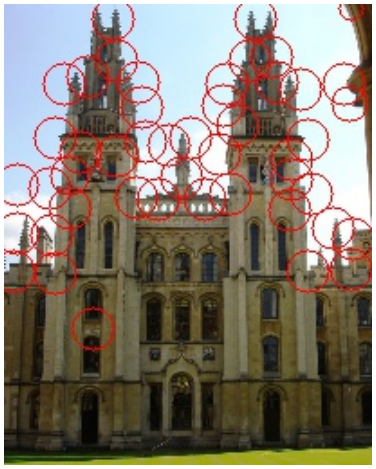


(a) Blobs detected for all_souls_000026.jpg

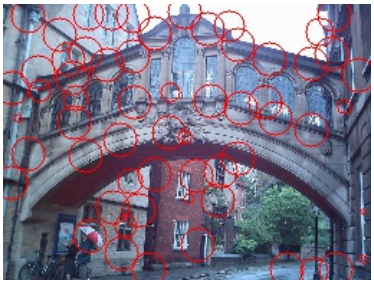


(b) Blobs for oxford_000317.jpg

Figure 1: Visualizing our algorithm for blob detection. We see that corners are prominently detected as they contain the most useful information.



(a) SURF keypoints detected for all_souls_000026.jpg



(b) SURF keypoints for oxford_000317.jpg

Figure 2: Visualizing our algorithm for plotting keypoints obtained over various scales of sigma $\sigma = [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6]$. This helps the algorithm to be robust to scale variations. We chose the same images as above to visualize to notice the difference between the two algorithms. Although, there are slight changes in the way we reduce overlaps amongst keypoints, we observe that both algorithms are quite good at detecting corners as local features, hence give rise to good descriptors.