MCA Assignment 1 Report

Vishaal Udandarao
2016119

February 2020

1 Colour Autocorrelogram

- **Pre-processing**: Every image is downsampled to 100x100 dimensionality. Now, we require a set of discrete representative colours which are common to any given set of images. For this, I first consider every colour in the RGB cube (256x256x256) and then discretize/bin this space into a representative space of dimensionality 64x64x64. This is done by a simple mapping wherein every pixel p is quantised by the formula: floor(p/4). To get a much more lower dimensional set of representative colours, I apply K-means clustering on this reduced 64x64x64 colour space with a K=64. Now, we are finally equipped with a set of 64 colours which are then used to compute the colour histogram for every image. To process every image, we first quantise the image into the discretized space (64x64x64) by applying the above functional transformation (floor(p/64)). We then assign each pixel to one of the 64 representative colours retrieved by K-means clustering. Hence, every image can then be compactly represented by a set of 64 colours.

- **Algorithm details**: The algorithm consists of recovering the probabilities of pixel proximities with respect to a discrete set of colours. In our case, we consider m=64 to be the number of discrete colours considered (as explained in the section above). We also select a set of 4 proximity distances (1, 3, 5, 7) at which to compute the probability values. Therefore for every image, our image representation will be of dimensionality 4x64 (where 4 is the number of distances considered and 64 is the number of discrete colours).

- **Results**: I have considered precision values at different lengths of top retrieval results. The different lengths are [100, 200, 300, 400, 500, 600, 700, 800, 900]. The following are the empirical results obtained:

    1. Maximum precision: 0.082 (Obtained for the query image: radcliffe_camera_000523.jpg)

2. Minimum precision: 0 (Obtained for the query images: cornmarket_000047.jpg, oxford_000317.jpg, cornmarket_000105.jpg, oxford_002416.jpg, cornmarket_000019.jpg and pitt_rivers_000119.jpg)
3. Average precision: 0.0128
4. Recall: 0.063
5. F1-score: 0.0213
6. Average time required to perform a retrieval: 1.783s
7. Average percentage of good queries retrieved: 8.67%
8. Average percentage of ok queries retrieved: 12.83%
9. Average percentage of good queries retrieved: 17.68%

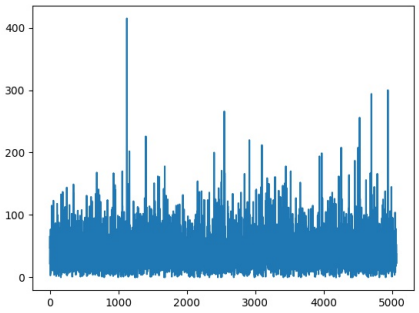
- **Analysis**: The results of the retrieval performed by using the colour autocorrelogram are quite poor. This might be possible due to a multitude of reasons including:

  1. Firstly, the discretization of the 256x256x256 space into a much lower dimensional 64x64x64 space might be leading to a bottleneck for the colour representations. By directly binning the RGB cube into a smaller space, we are constraining the colour space to have a much poorer discrimination between colours.
  2. Secondly, we only take 64 discrete colours as our colour set. This vastly diminishes the representational capacity of the feature extraction algorithm by pushing even very distinct colours into the same colour clusters. Therefore, albeit K-means performs a pure clustering, the colour clusters obtained may not be very pure. This could lead to a major drop in performance.
  3. The downsampling of images could be another major factor in the low performance. By downsampling the images across the entire dataset to a 100x100 dimensionality, we are not preserving the finer details within the image, which may be quite important for richer representations. By downsampling, we are essentially compressing multiple pixels into one pixel which can lead to loss in information that can be leveraged by the colour autocorrelogram algorithm.

  We also see that primarily images from the oxford_*.jpg class and cornmarket_*.jpg class have the poorest results. On looking at the images, we clearly see that most of the images in the oxford_*.jpg class are very generic (bridges, front facing buildings, castles etc). Hence, the retrieved images are also from a wide range of classes due to this generalization. Similarly, the images from the cornmarket_*.jpg class which have poor retrieval results are images of cottages which are also present in large numbers in the other classes in the dataset. A few examples of such images from both the classes are:

2 LoG Blob Detection

- **Pre-processing**: Every image is downsampled to 100x100 dimensions to process the features faster. The images are also converted to grayscale and then scaled by using min-max normalization (every pixel intensity value is divided by 255).
- **Algorithm details**: The algorithm first creates a scale space by considering an incremental range of σ (standard deviation for the Gaussian filter) values. For this implementation, I have taken 5 different σ values in the range ([1.35, 1.674, 2.075, 2.573, 3.190]). For each scale value, the image is convolved with an LoG (Laplacian of Gaussian at scale σ) filter whose responses are considered as the potential high pixel intensity blobs.
- **Analysis**: The following are the experimental results obtained:
  1. Average number of blobs per image: 40.62
  2. Max number of blobs per image: 415
  3. Min number of blobs per image: 0
  4. Total time for blob detection: 2119.27s (35.32 mins)
  5. Average time for blob detection per image: 0.41s
  6. The distribution of the number of blobs over the entire training set is plotted below:

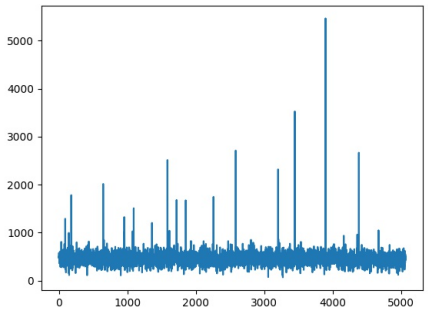


- **Visualization**: A few examples of blobs detected by the LOG detection method are shown below:

3 SURF Blob Detection

- **Pre-processing**: Every image is downsampled to 100x100 dimensions to process the features faster. The images are also converted to grayscale and then scaled by using min-max normalization (every pixel intensity value is divided by 255).
- **Algorithm details**: The algorithm first computes the integral image for every pre-processed image, and then the determinant of the hessian matrix of the box-filtered integral image is taken as the potential high pixel intensity blobs. Non-max suppression is then performed over these potential blobs. I have used 5 different scale levels (5 distinct σ values). The σ values taken are [1.35, 1.674, 2.075, 2.573, 3.190]. These were optimally found out after doing a hyperparameter tuning. Taking too low of an initial sigma (<1) turned out to be an overkill since it detected too many local features.
- **Analysis**: The following are the experimental results obtained:
  1. Average number of blobs per image: 465.07
  2. Max number of blobs per image: 5465

3. Min number of blobs per image: 69
4. Total time for blob detection: 261.26s (4.35 mins)
5. Average time for blob detection per image: 0.05s
6. The distribution of the number of blobs over the entire training set is plotted below:



- **Visualization**: A few of the blobs extracted by SURF detection are: