EEE3032
Computer Vision and Pattern Recognition
Coursework Assignment
Visual Search of an Image Collection

- Add failure case results in appendix
- Explain theory of P-R graphs
- Confusion matrix
- Explain how our categories are level 2, but algs are level 1

Joshua Tyler

April 12, 2016

Abstract

Write this at the end!

Contents

1 Introduction

Large databases of images exist in various fields. Indexing ans searching these using textual descriptors is generally not effective [1, p.657]. Visual search offers an alternative to this, by allowing a user to search an image collection using an image as the query, with the aim being to return images similar to the query. This is achieved by generating and comparing numeric, vector based, image descriptors which can be treated similar to words.

This report presents the implementation of such a system in matrix laboratory (MATLAB). Several different methodologies will be considered, and their results compared and contrasted.

In order to test the system, the Microsoft Research Cambridge Object Recognition Image Database, version 2.0 will be used. This is a freely available image database which is split into 20 rows, with each row containing similar images [2].

1.1 System Design

A common framework was used to apply generate descriptors from all of the images, compare them, and display results. This framework uses a MATLAB function handle to input both the descriptor and distance measure. This method allows the core code-body to be generic, so that in order to run a different test only the external descriptor and/or distance measure functions need to be changed.

A block diagram of the operation of the system is shown in Figure 1.1.

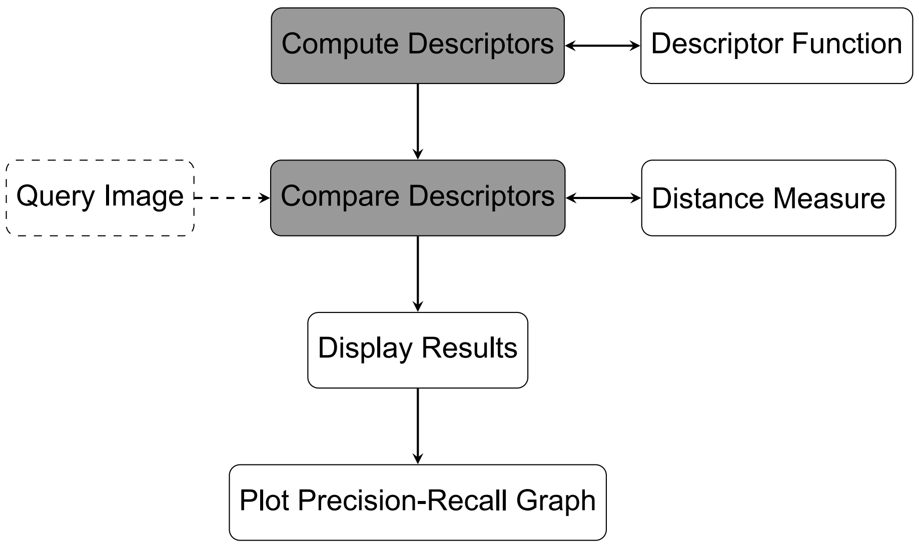


Figure 1.1: System design

1.1.1 Test Methodology

In order to test the system, a unit test has been written for all of the descriptor and distance measure functions. Each of these tests complies with MATLAB's unit testing framework, so can be executed using the `runtests` command.

Test pattern generation

Some tests require test patterns in order to test their functionality correctly. For this purpose a suite of test pattern generation programs was written to create image which contain:

- A uniform colour.
- A uniform pattern.

- A combination of the above.
- A grid of several of the above images.

These test patterns can be used both internally to the unit tests, as well as to perform system level testing.

1.2 Techniques Implemented

This section discusses the visual search techniques implemented, along with the theory of their operation and the difficulties which were encountered whilst implementing them.

1.2.1 Global Colour Histogram

A global colour histogram quantises the RGB space into a series of bins. There are Q bins for each dimension, and therefore Q^3 bins for the total image. Each pixel in the image is then sorted into the appropriate bin, and the sum of the bin totals is normalised to 1. This vector histogram, of length Q^3, can therefore be used to describe the image.

The implementation of this function was relatively straightforward, since example code for the function had been provided [3]. The only modifications made to the code were to explicitly state the histogram bin edges. This was necessary to provide a robust descriptor since the provided code normalised the histogram to the range of the input. Therefore an image containing only values in one area of the RGB space, would appear similar to an image with uniform distribution.

The unit tests for this function consist of generating images of known RGB distribution, calculating the RGB histogram by hand for these images, then comparing that to the output of the function.

1.2.2 Texture Histogram

Texture is a concept in computer vision, but is difficult to define. Texture loosely corresponds to a repeating pattern of edge characteristics, examples of which would include grass pebbles, and hair [1, p. 194]. Texture is important because it appears to be a strong indicator of object identity.

The texture histogram function is implemented as the concatenation of two functions: an edge orientation function, and a histogram function. The edge orientation function returns a two dimensional array where each element contains a value corresponding to an estimate of the edge orientation at that location. This estimate is in the range $-\pi$ to $+\pi$, but is normalised to 0 to 1. Weak edges are set to a normalised value of 0.5, corresponding to 0 rad.

In order to perform the edge detection, and edge angle estimation, the function uses the Sobel operator [4]. The sobel operator is formed of two kernels which are convoluted with input image in order to perform edge detection in the x and y directions respectively. The kernels are defined in Equation 1.1.

$$K_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1.1}$$

The edge magnitude and angle are calculated as described in Equations 1.2 and 1.3 respectively. It should be noted that the operates in these equations apply on an element-by-element basis, not to the array as a whole. In addition, the K_y value is inverted in order to allow the performance to match the MATLAB library function `imgradient`. This allows for easier testing. The pixels with edge magnitudes above a certain threshold are passed to the histogram generation function.

$$\text{Mag} = \sqrt{{K_x}^2 + {K_y}^2} \tag{1.2}$$

$$\theta = \text{atan2}(-K_y, K_x) \tag{1.3}$$

In order to test the edge detection function, the output is compared to the MATLAB function `imgradient`, for several test images, plus a real image. In addition system level testing has been performed with several test images.

1.2.3 Gridding and Concatenation of Features

Gridding of features allows descriptors to apply be considered locally to parts of an image. The implementation of this is fairly simple in MATLAB. The image can be split using the native array indexing of MATLAB to select a subset of the image array. The descriptor function can then be applied to each of these sub images, and the resultant descriptors concatinated into a matrix.

The gridding functon is tested by feeding a gridded sample image into the function and comparing the output to a manually calculated version of what is expected. A test descriptor function, which returns the image reshaped into a vector array is used to simplify this process.

1.2.4 Distance Measures

In order to calculate the difference between descriptors, the L2 norm is used. This distance measure returns the square root of sum of the squares of the differences between elements of a descriptor, as defined in Equation 1.4, where d_i refers to individual differences between array elements.

$$\text{Distance} = \sqrt{\Sigma({d_i}^2)} \tag{1.4}$$

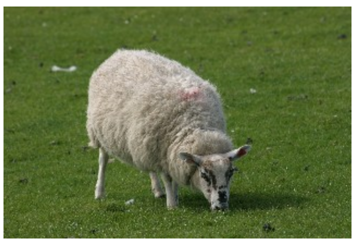
The L2 norm calculation function is tested by comparing the output of the function to manually calculated values for various test vectors.

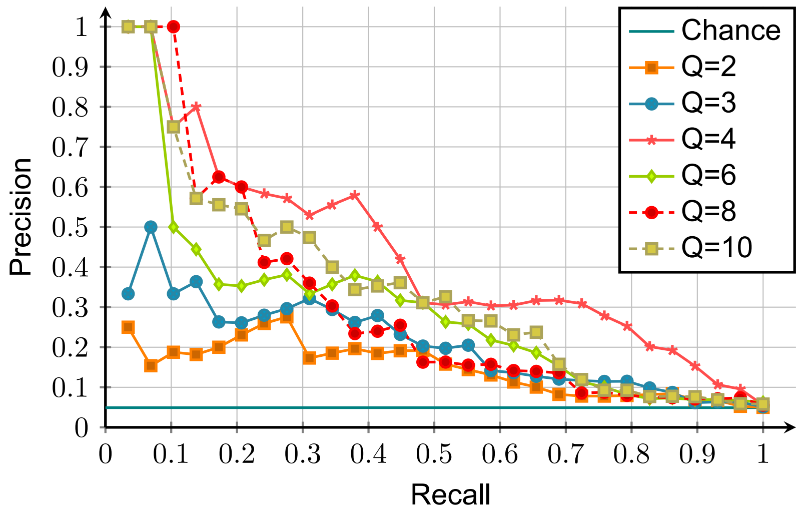
2 Experimental Results

2.1 Global Colour Histogram

The global colour histogram function is implemented as `H = vs_compute_rgb_histogram(img, Q)` where `H` is the descriptor returned, `img` is the image input, and `Q` is the quantisation level for each image dimension. The descriptor is therefore of length Q^3, as discussed in Section 1.2.1.

In order to analyse the effect of quantisation level on performance, descriptors have been generated for various values of Q, and the performance of each value analysed for each test value. Two test images have been chosen, 9_23_s and 13_1_s, and their results are displayed in Figures 2.1 and 2.2b respectively. These images have been chosen because they represent two contrasting examples of what could be presented to the algorithm. Image 9_23_s shows a sheep in a field. This image has two strongly dominant colours: the green of the grass, and the cream of the sheep's wool, and very little other colour content in the picture. On the other hand 13_1_s shows a collection of differently coloured books on a shelf. This image contains many different colours.

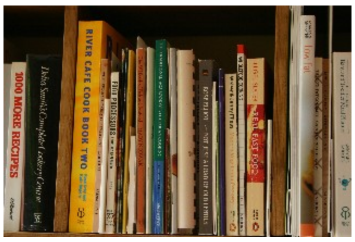

(a) Query image

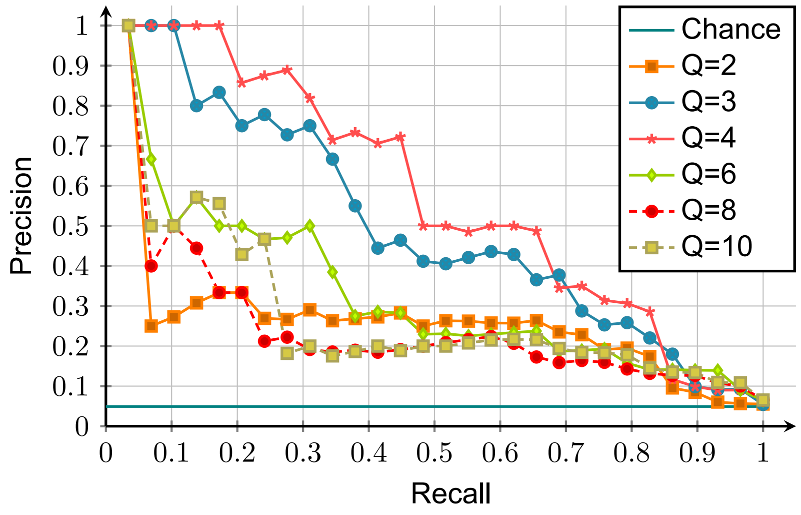


(b) Resultant graph

Figure 2.1: Global colour histogram results for query image 9_23_s

The results shown in Figures 2.1 and 2.2b match expectations suggesting that a very low quantisation value does not deliver good results, because there are not enough bins to differentiate between the images. In addition very high values, such as $Q = 10$ do not work well because there is two many possible bins, and as such nominally similar colours will be sorted into different bins, and so will not be considered similar by the distance measure function. Both test images suggest that a quantisation level of $Q = 4$, yielding 64 bins, gives the best result, since the line joining these data points is closest to the top right hand corner.


(a) Query image



(b) Resultant graph

Figure 2.2: Global colour histogram results for query image 13_1_s

2.2 Gridding

Gridding is implemented as the function `F = vs_grid(img, h_level, v_level, compute_function)`, where `img` is the image input, `h_level` is the horizontal quantisation level, `v_level` is the vertical quantisation level and `compute_function` is the descriptor generation function. This configuration therefore gives us two independent variables, `h_level` and `v_level`.

However, since most of the images in the dataset have a quantisation level of approximately 1.5:1, it makes sense to set the vertical quantisation level to 1.5 times the horizontal quantisation level. Therefore the overall quantisation level, Q can be defined, where `h_level = ceil(1.5*Q)`, and `v_level = Q`.

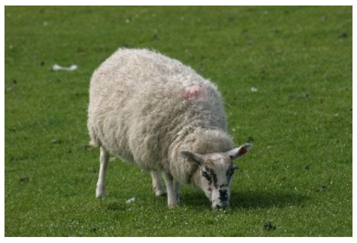
In a similar manner to that of Section 2.1 the optimum quantisation factor will be determined by computing descriptors for various values of Q and then, using two test images, determining which has best results. For this test, the value of Q for the global colour histogram which prodced the best results ($Q = 4$) will be used. The same test images as used in Section 2.1 have also been maintained in order to allow a performance comparison with the non-gridded image.

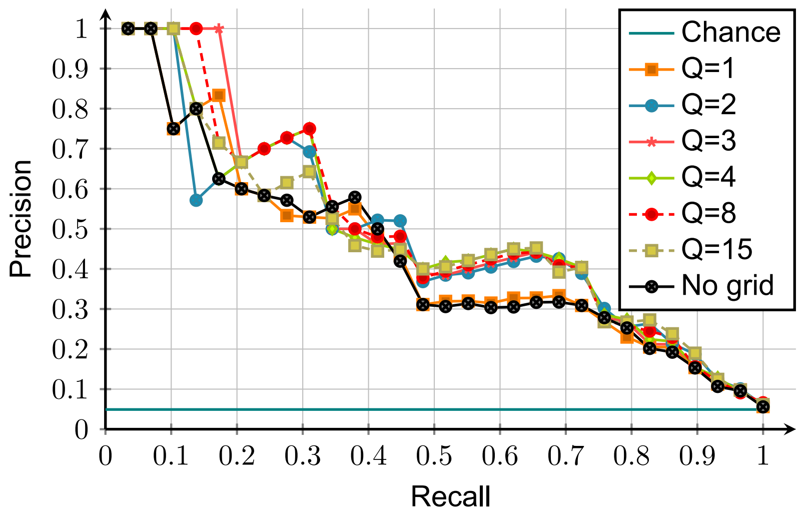
The same effect of increasing Quantisation factor as experienced with the global colour histogram is expected. Increasing the value should increase performance up to a maximum, at this point the performance should decrease with increasing quantisation factor. One reason for this is that tighter grids mean that small movements in the object could potentially result in the image being quantised in a very different way, meaning visually similar images would be distant from each other in the quantised space.

Figures 2.3 and 2.4 show the results of the gridding process.

Figure 2.3 shows little variation across gridding quantisation levels, this is likely due to the fact that the image has only two dominant colours, so all grids have similar colour histograms. Gridding does in the general case show slight performance improvement with this image however. Notably increasing prevision values for higher quantisation levels.

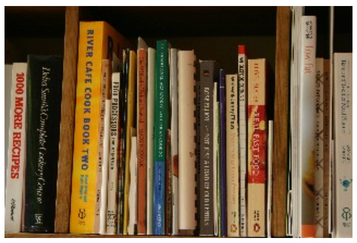
Figure 2.4 on the other hand shows great variation in the result with grid size. Increasing the grid quantisation level to 2 greatly increases the performance up to a recall of around 0.4, and a grid quantisation level of 1 also increases the performance by a large amount also. In addition to this, much higher quantisation steps, such as 8 and 15 show greatly decreased performance. This is due to reasons mentioned previously.

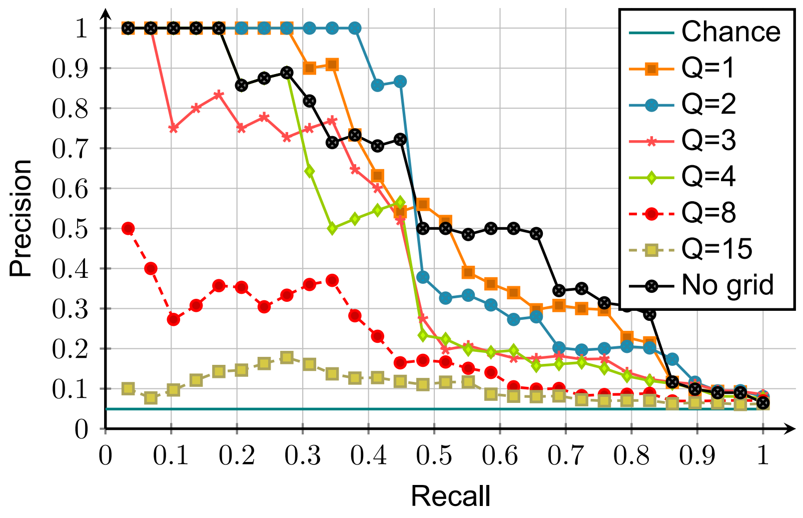

(a) Query image



(b) Resultant graph

Figure 2.3: Gridded global colour histogram results for query image 9_23_s


(a) Query image



(b) Resultant graph

Figure 2.4: Gridded global colour histogram results for query image 13_1_s

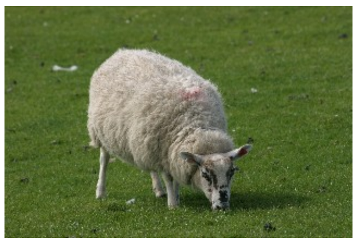
2.3 Texture Histogram

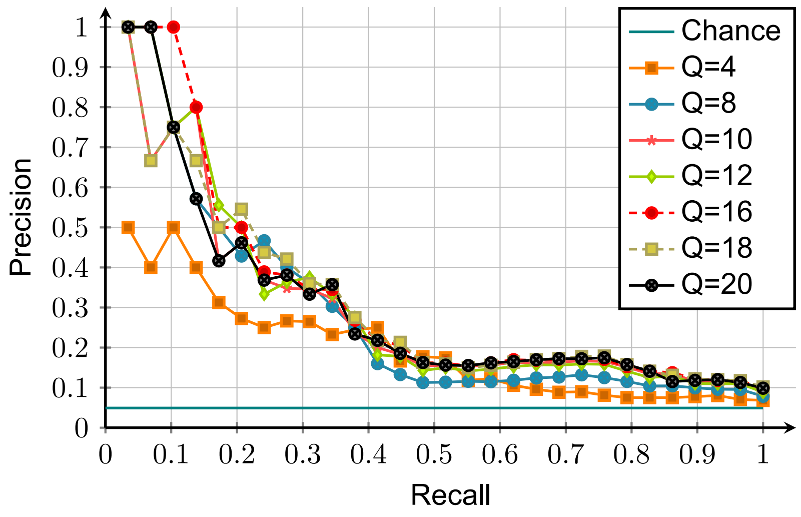
The edge orientation function is implemented as `F = vs_edge_detect(img, compute_function, strength)` where `F` is the descriptor returned, `img` is the image input, `compute_function` is the histogram generation function, and `strength` is the minimum edge strength to consider (range 0–1). The histogram generation function is `H = vs_compute_histogram(img, Q)` where `H` is the histogram returned, `img` is the image input and `Q` is the quantisation factor. When `vs_compute_histogram` is used as the `compute_function` in `vs_edge_detect`, the system can produce an edge orientation histogram, with two independent parameters. These parameters determine the minimum strength edge to consider, as well as the quantisation level of the histogram. These parameters will be referred to as E and Q respectively.

Additionally, since texture is an inherently local feature, the image will be gridded. The descriptor function handle is therefore : `@(x)vs_grid(x,3,2, @(x)vs_edge_detect(x, @(x)vs_compute_histogram(x,Q),`
The grid quantisation factor of $Q = 2$ is used, since this was determined to be the most effective for gridding the RGB histogram in Section 2.2.

Evaluating the effect of changing both E and Q together is computationally expesive, and the results are hard to visualise, since a 3 axis plot would be required. Therefore the two variables will be evaluated separately.

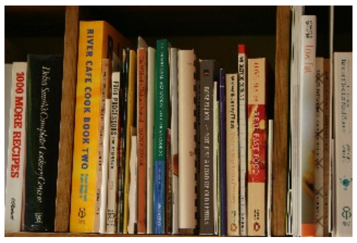
Initially the effect of changing the histogram quantisation level, Q, will be investigated using a constant E of 0. This will leave all edges in the image for the consideration of the histogram

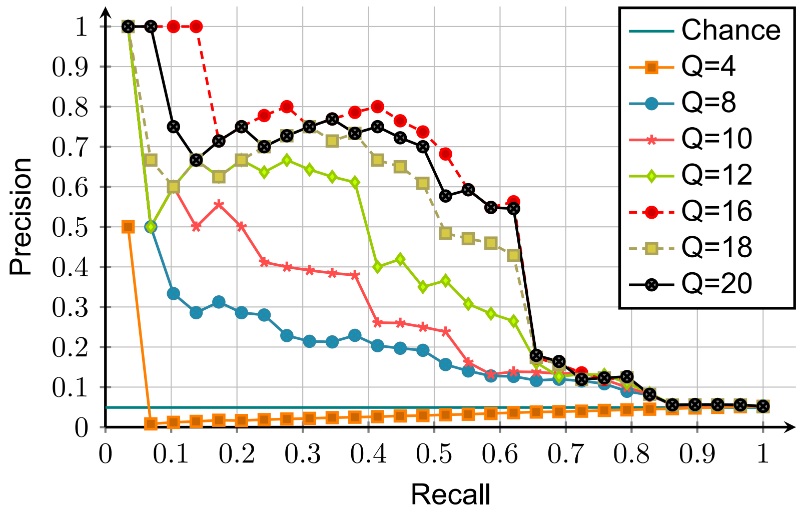

(a) Query image



(b) Resultant graph

Figure 2.5: Gridded texture histogram results for query image 9_23_s, with $E = 0$


(a) Query image

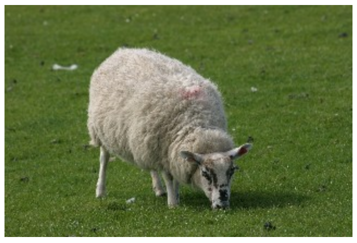


(b) Resultant graph

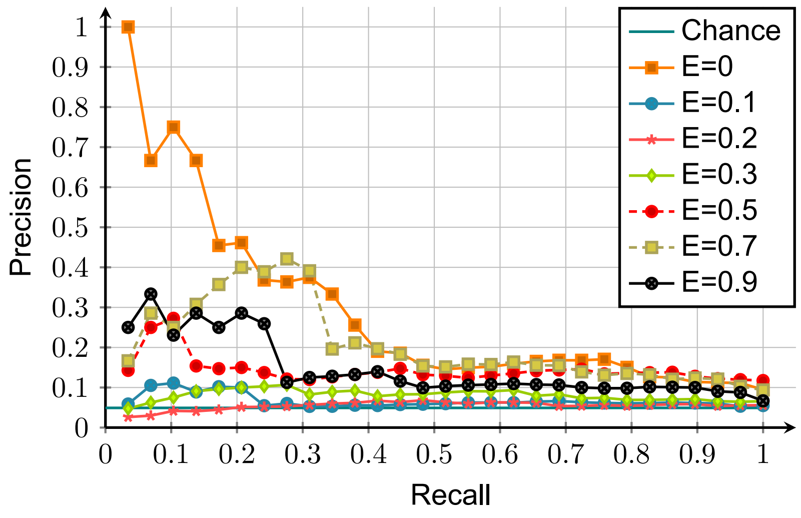
Figure 2.6: Gridded texture histogram results for query image 13_1_s, with $E = 0$

Figure 2.5 shows that for the sheep image, there is very little difference in performance with quantisation levels greater than approximately 4. Figure 2.6 shows that the books, on the other hand, performance is dependant on quantisation level. Higher quantisation levels continue to improve performance, up to approximately $Q = 16$, after this point performance decreases.

2.4 Concatenation of Descriptors

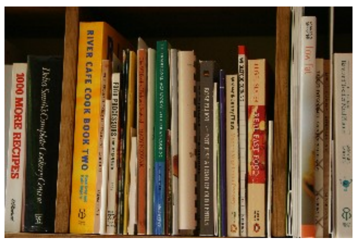


(a) Query image

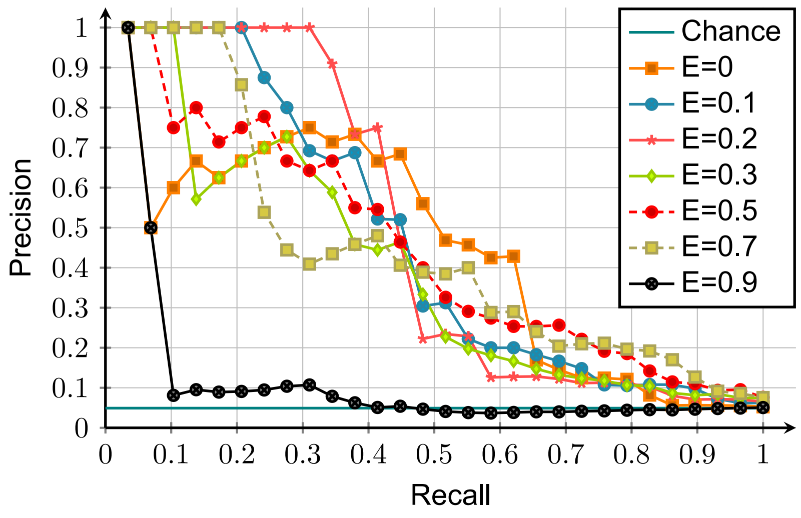


(b) Resultant graph

Figure 2.7: Gridded texture histogram results for query image 9_23_s, with $Q = 16$



(a) Query image



(b) Resultant graph

Figure 2.8: Gridded texture histogram results for query image 13_1_s, with $Q = 16$

3 Conclusion

Bibliography

[1] D. Forsyth and J. Ponce, *Computer Vision. A modern approach*, 2nd ed. Pearson, 2012.

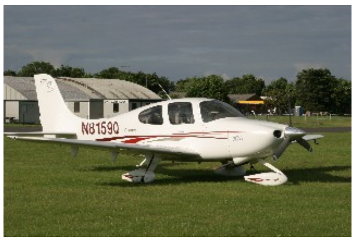
[2] A. Criminisi, “Microsoft Research Cambridge Object Recognition Image Database, version 2.0,” http://research.microsoft.com/en-us/projects/objectclassrecognition/, 2004, [Online; accessed 09-April-2016].

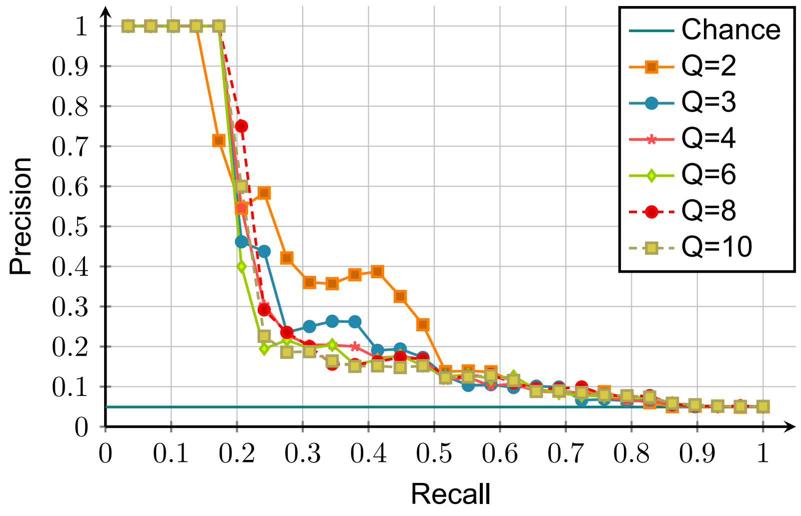
[3] J. Collomosse, “Lecture notes for EEE3032 – Computer Vision and Pattern Recognition,” 2016.

[4] I. Sobel, “History and Definition of the so-called “Sobel Operator”,” https://www.researchgate.net/publication/239398674_An_Isotropic_3_3_Image_Gradient_Operator, 2015, [Online; accessed 10-April-2016].

A Foo

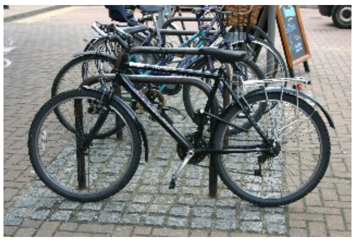
Placeholder

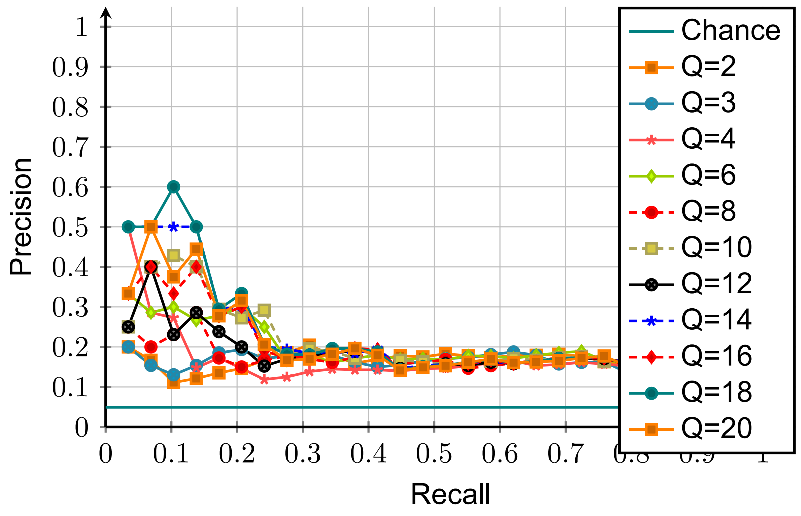

(a) Query image



(b) Resultant graph

Figure A.1: Gridded global colour histogram results for query image 4_12_s


(a) Query image



(b) Resultant graph

Figure A.2: Global texture histogram (NO GRIDDING) results for query image 8_7_s